Impresión en consola	
System.out.**print**(""); System.out.**println**("");	Imprime sin salto de línea y con salto de línea
System.out.**printf** ("Cadena formato %d" , argument);	Imprime con formato
System.out.**printf**("%1.2f", 12500045465.445645665);	12500045465,45

Enteros	**byte**	**short**	**int**	**long (L)**
Decimales	**float (F)**	**double**		
Otros	**char**	**boolean**		

Comentarios	**/*** … diversas líneas …***/**	**/**** … documentación …***/**	**//**… una sola línea

Variables constantes	**final** double PI **=** 3.14159265358979323846264338327 95;

*** , / , % , + , -**	Operadores aritméticos	a * b , a / b , a % b , a + b , a - b
= , += , -= , *= , /= , %=	Operadores de asignación	a = b , a += b , a -= b , a *= b , a / b , a % b (a=a % b)
== , != , < , > , <= , >=	Operadores de comparación	a == b , a != b , a < b , a > b , a <= b , a >= b
++ , -- , (tipo) expr	Operadores especiales	a++ , a-- (post); ++a , --a (pre) , a = (int) b
&& , || , !	True = si las 2 expresiones son true , True = si una expresión es true , Lo contrario que	

Clase Scanner - (import java.util.Scanner) - (métodos no estáticos)		
nextInt	int i = scr**.nextInt()**;	int var1 **= Scanner** scr **= new Scanner(System.in);** scr**.nextInt();**
nextDouble	double d = scr**.nextDouble()**;	
nextLine	String str = scr**.nextLine();**	// Vacía buffer entrada teclado antes de usar nextLine scr**.nextLine();**

Clase GregorianCalendar - (import java.util.GregorianCalendar)		
DATE	25	**GregorianCalendar** fecha **= new GregorianCalendar();**
MONTH	1	int dia **=** fecha**.get(GregorianCalendar.DATE);**
YEAR	2016	System.out.println**(**dia + "/" + (mes + 1) + "/" + año**);**

Clase Date - (import java.util.Date) - representa un instante específico en el tiempo	
Date fecha = new Date();	
Date fechaContrato = calendar.getTime();	//calendar es un objeto GregorianCalendar

Convertir un valor String a numérico	**Convertir un valor numérico a String**
int valorEntero = **Integer.parseInt(**cadena**);**	**String numero = String.valueOf (1234);**
double valorReal = **Double.parseDouble(**cadena**);**	**String fecha = String.valueOf (new Date());**
	String cadena **= Integer.toString(**valorEntero**);**

Clase Random - (import java.util.Random;) - Genera números aleatorios	
Random rnd = new Random();	//Creamos un objeto Random
int numero = rnd.nextInt(1001);	//devuelve entre 0-1000
Int ranDate = rnd.seSeed(new Date().getTime());	//devuelve números aleatorios a partir de la hora

Clase Math			
abs	Valor absoluto	int x = **Math.abs(-2)**;	x = 2;
sqrt	Raíz cuadrada	double x = **Math.sqrt(9)**;	x = 3.0;
pow	Potencia	double x = **Math.pow(2, 3)**;	x = 8.0;
PI	Número PI	double x=**Math.PI**	x = 3.14159265358979;
round	Redondea decimal a entero	int x = (int)**Math.round(4.6)**;	x = 5;
max	Devuelve el mayor	int x = **Math.max (5, 10)**;	x = 10;
min	Devuelve el menor	int x = **Math.min(5, 10)**;	x = 5;
random	Nº real aleatorio [0-1]	double x = **Math.random()**;	x = 0.20614522323378;
	Nº entero aleatorio 1-10	int x = (int)(**Math.random()** * (Mx-Min+1) + Min);	//devuelve entre 1-6

Clase String - Texto entre comillas			
equals	Compara 2 cads.	boolean var = s1**.equals(**s2**);**	true o false;
equalsIgCa	Compara 2 cads.	boolean var = s1**.equalsIgnoreCase(**s2**);**	true o false;
length	Longitud de cadena	s1 = "Prueba"; int x = s1**.length()**;	x = 6;
charAt	Guarda char de pos pedida	s1 = "Prueba"; **char** x = s1**.charAt(2)** ;	x ='u';
substring	Genera subcadena	s1 = "Ey tu"; String x = s1**.substring(**0,2**)**;	x ="Ey";
toCharArra	String a array	**char []** x **=** s1**.toCharArray();**	x[0]= 'H'; x[1]= 'o';…
indexOf	1era pos donde aparece una cadena de texto	String s1 = "Quería decirte que"; int x **=** s1**.indexOf(**"t"**);**	x=12; Devuelve -1 si no la encuentra
lastIndexOf	Última pos donde aparece una cadena de texto	String s1 = "Quería decirte que"; int x **=** s1**.lastIndexOf(**"ue"**);**	x=16; Devuelve -1 si no la encuentra
endsWith	True si cadena termina con texto determinado texto	String s1 = "Quiero que te vayas"; boolean x **=** s1**.endsWith(**"vayas"**);**	x = true;
startsWith	True si cadena empieza con texto determinado	String s1 = "Quiero que te vayas"; boolean x **=** s1**.startsWith(**"Qui"**);**	x = true;
replace	Sustituye en cadena las apariciones de un carácter	String s1 = "Mariposa"; String x **=** s1**.replace(**'a'**,** 'e'**);**	x= "Meripose"
replaceAll	Sustituye en cadena las apariciones de una cadena	String s1="Armadillos"; String x **=** s1**.replaceAll(**"Ar"**,**"Er"**);**	x="Ermadillos";
toUpperCa	Cadena a mayús.	String s1="Hola"; String x **=** s1**.toUpperCase();**	x= "HOLA"
toLowerCa	Cadena a minús.	String s1="Hola"; String x **=** s1**.toLowerCase();**	x= "hola"
contains	True si encuentra cadena	Boolean x = s1**.contains(**"a"**);**	s1 = "hola"; x = true;
isEmpty	True si es igual a 0	Boolean x = s1**.isEmpty();**	x = false;
Character.toUpperCase(caracter**);**		Convierte el carácter indicado en un carácter en mayúsculas	

Clase StringBuilder - (StringBuilder sb = new StringBuilder();) - Texto entre comillas			
lenght	Devuelve longitud cadena	sb**.lenght()**;	
capacity	Devuelve capacidad cadena	sb**.capacity()**;	
insert	Añade caracteres principio	sb**.insert(**pos,String**)**;	
append	Añade caracteres al final	sb**.append(**String**)**;	
append	Invierte dirección de cadena	sb**.reverse()**;	
Delete replace	Modifican la cadena	sb**.delete(**posInicial,posFinal**)**; sb**.replace(**posInicial,posFinal,String**)**;	

Enumerados	**public enum** DiaSemana **{**v1**,** v2**,** v3**,** v4**,** v5**,** v6**,** v7**}** DiaSemana primerDia **=** DiaSemana.v1**;**	generar una lista de valores

Array unidimensional (vectores) - Almacena tipos primitivos y objetos	
int **[]** miArray **= new** int **[**10**];**	//declarar el array
int [] miArray **= {**15, 25, …, n**};**	//declarar e inicializar el array
miArray**.length;**	//devuelve longitud del array
miArray **=** cadena**.getBytes();**	//transforma cadena en array de bytes

Array multidimensional (matrices) - Almacena tipos primitivos y objetos	
int **[] []** miArray **= new** int **[**5**] [**5**];**	//declarar el array
int **[][]** miArray **= {{**15, 25**},{**50, 60**},{**Vn, Vn**}};**	//declarar e inicializar el array

Estructuras de decisión: If…else, If…else if - Se usan para ejecutar 2 trozos de código mutuamente excluyentes.

		F. clásica	F. compacta

if (condicion**) {**
sentencia 1n;
} else {
sentencia 2n;
}
	if (condicion**) {**
	sentencia 11;
	} else if { (condición1**)**
	sentencia 11;
	} else {…**}**
		if (condicion**)**
		variable = v1;
		else
		variable = v2;
			variable = condicion ? v1 : v2 ;

[switch…case]	**while (**condición**) {**	**While:**
Se usa cuando se conoce la lista valores (int, boolean y char).	instrucciones si es true; **}**	Se ejecuta 0 o más veces.
switch (expresión**) {** **case 1: case 2:** instrucciones; [**break;**] **default:** instrucciones; **}**	**do {** instrucciones; **} while (**condición**);**	**Do While:** Se ejecuta al menos una vez.
	for (inicBucle; condición; contBucle) { instrucciones; **}**	**For:** Se usa cuando conocemos el número de veces que se repite la ejecución de un código.
for (int dato **:** miArray**) {** System.out.println(dato); **}**		**For - Each**: se usa pare recorrer arrays de forma sencilla.

Crear clase	**[public/abstract/final/..] class** NomClase **{**
	atributos;
	métodos;
	}
	
	this.nomAtributo**;**
	this.nomMetodo**();**
	this();
		//una sola clase public en cada fichero
		
		Modificadores: public, private , [de paquete]
		Pueden ser: [e objeto] o static
		
		//referencia a un atributo del objeto actual
		//llamada a un método para el objeto actual
		//llamada a un constructor desde otro constru.

	private int valor1**;**
	private static int valor2**;**
	private final String valor3**;**
		//declarar atributos: objeto, de clase o static
		//compartida por todos los objetos de la clase
		//final prohíbe cambiar el valor de la variable

	public NomClase **(**int valor**) {** **this.**valor1 **=** valor**; }**	//constructor //mismo nombre que la clase
	public int getNomMetodo**() {** **return** this.valor1; }	//método GET, si devuelve: se le llama función //return para devolver dato
	public void setNomMetodo**(**int valor**) {** **this.**valor1 **=** valor**;**	//método SET, //no devuelve valor: se le llama procedimiento
	public static tipo nomMetodo **(**int valor**) { …}** **;**	//método estático: no opera sobre objeto. //llamar a método estático: Clase.método

Instancia clase	**NomClase** obj1 **= new NomClase();**
	obj1**.nomAtributo;**
	obj1**.nomMetodo();**
	NomClase.nomMetodo();
		//instancia, llamada al constructor
		//accede a un atributo si es public
		//accede a un método del objeto (dinámico)
		//accede a un método de la clase (estático)

Object - toString()	
@Override public String **toString()** { **return** "Velocidad: " + velocidad + " / " + nombre**;**	//sobrescribe el método toString()

ArrayList - (import java.util.ArrayList)	

ArrayList<NomClase**>** miArrayList **= new ArrayList();**
ArrayList<NomClase**>** miArrayList **= new ArrayList(**10**);**
ArrayList<NomClase**>** miArrayList **= new ArrayList(**Collection c**);**
	//declarar un ArrayList vacío
	//declarar un ArrayList con 10 posiciones
	//declarar ArrayList a partir de colección

miArrayList**.add(**"España"**);** miArrayList**.add(**1 **,** "Italia"**);**	//inserta elemento en última posición //inserta elemento en posición indicada
miArrayList**.get(**2**);**	//devuelve elemento de posición indicada
miArrayList**.set(**1 **,** "Alemania"**);**	//modifica elemento almacenado
miArrayList**.indexOf(**"España"**);** miArrayList**.lastIndexOf(**"España"**);**	//devuelve pos primer elem. encontrado //devuelve pos último elem. encontrado
miArrayList**.size();**	//devuelve tamaño del ArrayList
miArrayList**.remove(**"España"**);** miArrayList**.remove(**1**);**	//elimina elemento determinado //elimina elemento de posición indicada
miArrayList**.clear();**	//borra todo el contenido del ArrayList
miArrayList**.clone();**	//devuelve una copia del ArrayList
miArrayList**.contains(**"España"**);**	//devuelve true si encuentra el elemento
miArrayList**.isEmpty();**	//devuelve true si el ArrayList está vacío
miArrayList**.toArray(**miArray**);**	//convierte ArrayList a un Array

LinkedList - (import java.util.LinkedList)	
LinkedList pila **= new LinkedList();**	//declarar pila
pila**.getFirst();** pila**.getLast();**	//devuelve el primer elemento //devuelve el último elemento
pila**.removeFirst();** pila**.removeLast();**	//elimina y devuelve el primer elemento //elimina y devuelve el último elemento
pila**.addFirst();** pila**.addLast();**	//añade al principio de la pila //añade al final de la pila
pila**.peek();**	//examina primer elem. de lista sin borrarlo

Iterator - (import java.util.Iterator) - Define objetos que permite recorrer los elementos de una colección	
Iterator<NomClase**>** miIterator **=** miArrayList**.iterator();**	//declarar un Iterator asociado a ArrayList
while (miIterator**.hashNext()) {** System.out.println(miIterator**.next()**);	//bucle para leer un Iterator
iter**.remove();**	//elimina el último elem. devuelto por next

Operaciones con Arrays y Colecciones - (import java.util.Arrays)	
Arrays.sort(miArray**);**	//ordena Array
Arrays.fill(miArray,elemento**);**	//rellena todo el array con el elemento
Arrays.binarySearch(miArray,elemento**);**	//búsqueda rápida en Arrays ordenados

Operaciones con Colecciones - (import java.util.Collections)	
Collections.fill(miArray,elemento**);**	//rellena todo la lista con el elemento
Collections.binarySearch(miArray,elemento**);**	//búsqueda rápida en lista ordenada
Collections.sort(miArrayList**);**	//ordena ArrayList
Establecer criterio de ordenación:	
Devuelve: 0 = indica que son iguales, 1 = indica que es mayor, -1 = indica que es menor	

class Articulo **implements Comparable<**Articulo**> {**
public int compareTo(Articulo o**) {..}**
}
	//paso1: implementar interfaz Comparable
	(**@Override**)
	//paso2: método compareTo

class comparaPorCriterio **implements Comparator<**Objeto**> {**
  public int compare(Objeto o1, Objeto o2**) {..]**
}
	//paso1: implementar interfaz Comparable
	(**@Override**)
	//paso2: método compare

Collections.sort(artículos**, new** comparaPorCriterio ()**);**	//invocar método para ordenar
Collections.reverse(miArray**);**	//invierte el orden de la lista

Excepciones -		

try,
catch,
finally
	try {
	} catch (Exception e) {
	} finally { … **}**
		//código susceptible de provocar una excepción
		//código que se ejecutará en caso de error
		//código que siempre se ejecutará

Propagar	void nomMetodo(args) **throws** Exception **{** … **}**	//controlar llamada al método con un try/catch
Lanzar	**throw new** nombreException**();**	//lanza excepción de Java o propia //lanza excepción de Java o propia con mensaje

Crear Exception propia	**class** MiError **extends Exception {**
	  public MiError(String msg) {
	  super(msg);
	  } **}**
	MiError error **= new MiError (**"Error"**);**
		//hereda de la clase Exception
		//código habitual para el constructor
		
		//instanciar a la clase creada.

Mensajes catch	System.out.println(**e**);
	System.out.println(e.**getMessage()**);
	e**.printStackTrace()**;
		//devuelve: java.lang.Exception: / by zero
		//devuelve: by zero
		//devuelve: java.lang. Exception: / by zero
		at Pruebas.main(Pruebas.java:12)

NumberFormatException	Error al convertir cadena a número
ArithmeticException	Error al dividir por cero
ArrayStoreException	Error por almacenar un objeto de otro tipo en un array
IllegalArgumentException	Error al pasar un argumento ilegal a un método
IndexOutOfBoundsException	Error que indica que el índice está fuera de rango
NegativeArraySizeException	Error si se crea un array con tamaño negativo
NullPointerException	Error al usar null donde se pide un objeto
InputMismatchException	Error de Scanner que indica que valor recuperado no coincide con el esperado

Herencia		Acceso subclase mismo paquete	Acceso subclase otro paquete
Acceso a atributos y métodos en subclase	**private** **sin especificar** **protected** **public**	No Si Si Si	No No Si Si
Upcasting	A una variable de tipo A, además de asignarle objetos A, se le pueden asignar objetos subclase de A.		
	class Persona **{ … }** **class** Alumno **extends** Persona **{ … }** Alumno alum **= new** Alumno**();** Persona pers **=** alum**;**	//una variable persona referencia a alumno	
Downcast	Una variable de una subclase referencia a un objeto de la superclase. Deshace upcasting previo, sino lanza excepción del tipo ClassCastException.		
	class Persona **{ … }** **class** Alumno **extends** Persona **{ … }** Persona pers **= new** Alumno**();** Alumno alum **= (**Alumno**)** pers**;**	//Deshace el Upcasting previo	

Heredar	**public class** NomClase1 **extends** NomClase2 **{}**
	super.nomMetodo()**;**
	super();
		//NomClas2 (subclase), NomClas1 (superclase)
		//llamada a método sobrescrito de la superclas
		//llamada al constructor de la superclase	

Instance of	pers **instance of** Persona**;**	//devuelve true si pertenece a la clase indicada	

Interface – Para especificar el comportamiento de clases		
Crear	public **interface** nomInable **{**	// métodos sin cuerpo y variables inicializadas
Utilizar	**class** NomClase **implements** nomInable **{**	//definir métodos con cuerpo
	class NomClase1 **extends** NomClase2 **implements** nomInable **{**	//todos los métodos abstractos de todas las interfaces deben ser implementados
Herencia	**public interface** nomInable3 **extends** nomInable1 **{**	//herencia entre interfaces

FileReader - (import java.io.*) - lectura de caracteres de manera secuencial	
FileReader fr **= new FileReader(**ruta**)	(**file**);**	***(FileNotFoundException)***
int caracter **=** fr**.read();**	***(IOException)*** //int con carácter leido
fr**.read(**miArray**);**	***(IOException)*** //lee caracteres a array
fr.**read(**miArray, posIni, cantidad**);**	***(IOException)*** //lee chars a array desde pos
fr**.close();**	***(IOException)*** //cierra el fichero abierto
BufferedReader - almacenamiento temporal de caracteres antes de la lectura	
BufferedReader br **= new BufferedReader(**fr**);**	
Lectura de caracteres como con FileReader	
br**.readLine();**	***(IOException)***//lee chars hasta null o salto línea

FileWriter - (import java.io.*) - escritura de caracteres de manera secuencial	
FileWriter fw **= new FileWriter(**ruta**);** **FileWriter** fw **= new FileWriter(**file, true**);**	***(IOException)*** //true indica que añade al final
fw**.append(**caracter**);**	***(IOException)*** //añade carácter al final fichero
fw**.write(**cadena**)	(**miArray**);**	***(IOException)*** //escribe en el fichero
fw**.write(**cadena, pIni, cant**)	(**miArray, pIni, cant**);**	***(IOException)*** //escribe n char desde pos
fw**.flush();**	***(IOException)*** //escribe sin cerrar el fichero
fw**.close();**	***(IOException)*** //cierra el flujo
BufferedWriter - almacenamiento temporal de caracteres antes de la escritura	
BufferedWriter bw **= new BufferedWriter(**fw**);**	
Escritura de datos como con FileWriter	
bw**.newLine();**	***(IOException)*** //equivale a salto de línea "\n"

FileInputStream - (import java.io.*) - lectura de bytes de manera secuencial	
FileInputStream fis = **new FileInputStream(**ruta**)	(**file**);**	***(FileNotFoundException)***
int leerByte = fis**.read();**	***(IOException)*** //int del carácter // -1 si no lee
fis**.read(**miArray byte**);**	***(IOException)*** //lee caracteres a array
fis**.read(**miArray byte, posIni, cantidad**);**	***(IOException)*** //lee chars a array desde pos
int bytesRestantes = fis**.available();**	***(IOException)*** //devuelve bytes por leer
fis**.close();**	***(IOException)*** //cierra el fichero abierto
DataInputStream - (import java.io.*) - adapta lectura de bytes a datos primitivos y Strings	
DataInputStream dis = **new DataInputStream(**fis**);**	
Lectura de bytes como con FileInputStream, esta clase dispone de métodos específicos para leer datos primitivos	
dis**.readXXX();**	***(IOException)*** //lee datos primitivos

FileOutputStream - (import java.io.*) - escritura de bytes de manera secuencial	
FileOutputStream fos = **new FileOutputStream (**ruta**);** **FileOutputStream** fos = **new FileOutputStream (**file, true**);**	***(FileNotFoundException)*** //true indica que se añade al final
fos**.write(**miArray byte**);** fos**.write(**miArray byte, posIni, numBytes**);**	***(IOException)*** //cadena.getBytes(); //escribe n caracteres de bytes desde pos
fos**.append(**char c**);**	//añade un carácter al final del fichero
fos**.flush();**	***(IOException)*** //escribe sin cerrar el fichero
fos**.close();**	***(IOException)*** //cierra el flujo
DataOutputStream - (import java.io.*) - adapta escritura de datos primitivos y Strings a bytes	
DataOutputStream dos = **new DataOutputStream(**fos**);**	
Escritura de bytes como con FileOutputStream, clase con métodos específicos para escribir datos primitivos	
dos.**writeBytes(**cadena**);**	***(IOException)*** //guarda cadena a fichero
dos.**writeXXX(**argumento**);**	***(IOException)*** //guarda datos primitivos

File - (import java.io.*) - permite el tratamiento de ficheros y directorios (borrar, renombrar, crear, …)	
File f = **new File(**file**);** **f.createNewFile();**	"..\\" //sube un nivel en la ruta de directorios ***(IOException)*** //crea el fichero indicado en f
f**.canRead();**	//devuelve true si se puede leer
f**.canWrite();**	//devuelve true si se puede escribir
f**.delete();**	//elimina el fichero, false si no puede
f**.exists();**	//true si existe el directorio o fichero
f**.isDirectory();**	//devuelve true si es directorio
f**.isFile();**	//devuelve true si es fichero
f**.length();**	//devuelve el tamaño en bytes del fichero
String[] miArray = f**.list();**	//devuelve nombre de ficheros - cmd dir
File[] miArray = f**.listFiles();**	//devuelve nombre y ruta de ficheros - dir
f**.mkdir();**	//crea el directorio si no existe, true si ok
f**.mkdirs();**	//crea directorios si no existen, true si ok
f**.renameTo(new File(**"\\ruta\\pruebas.txt"**));**	//cambia el nombre de fichero, true si ok
f**.lastModified();**	//devuelve milisegundos, transformar con Date

RandomAccessFile - (java.io.*) - acceso a cualquier posición de un fichero en cualquier momento	
RandomAccessFile raf = **new RandomAccessFile(**file, modo**);** **RandomAccessFile** raf = **new RandomAccessFile(**ruta, modo**);**	***(FileNotFoundException)*** //modo: "r" = read, "w" = escritura, "rw" =read/write
int caracter = raf**.read();**	***(IOException)*** //int del carácter leído // -1 si no lee
raf**.seek(**i**);**	***(IOException)*** //coloca puntero en pos indicada
raf**.getFilePointer();**	***(IOException)*** //devuelve la pos del puntero
raf**.length();**	***(IOException)*** //devuelve la longitud del fichero
raf**.readLine();**	***(IOException)***//lee chars hasta null o salto línea
raf**.readXXX();**	//devuelve dato primitivo
raf**.write(**caracter**);**	***(IOException)*** //escribe en fichero byte pasado
raf**.writeChars(**cadena**);**	***(IOException)*** //escribe dato primitivo
raf**.close();**	***(IOException)*** //cierra el fichero abierto

Serializable - guarda un objeto Java en una secuencia de bytes		
Serializar	**class** Persona **implements** Serializable **{ … }**	//de esta forma la clase es serializable
ObjectOutputStream - (import java.io.*) - dirige los objetos serializados a un fichero mediante FileOutputStream		
FileOutputStream f = **new FileOutputStream(**fichero**);**		***(FileNotFoundException)***
ObjectOutputStream oos = **new ObjectOutputStream(**f**);**		***(IOException)***
oos**.writeObject(**objeto**);**		***(IOException)*** //serializa objeto en fichero
oos**.writeXXX(**dato**);**		***(IOException)*** //serializa dato primi. en fichero
oos**.flush();**		//guarda en fichero el buffer
oos**.close();**		***(IOException)*** //cierra el flujo abierto
ObjectInputStream - (import java.io.*) - recupera objetos serializados en un fichero mediante FileInputStream		
FileInputStream fis = **new FileInputStream(**fichero**);**		***(FileNotFoundException)***
ObjectInputStream ois = **new ObjectInputStream(**fis**);**		***(IOException)***
(miArrayList<clase>**)** ois**.readObject();**		***(ClassNotFoundException)*** //deserializa objeto
ois**.readXXX();**		***(IOException)*** //deserializa dato primitivo
ois**.close();**		***(IOException)*** //cierra el flujo abierto