Clases internas o anidadas (1)

- ***Java* permite la creación de clases anidadas**. La utilidad de las clases anidadas permite mejorar la **estructuración** y la **legibilidad** del código.

- Para **instanciar** un nuevo objeto de una **clase anidada**, es **necesario hacerlo a través de un objeto de la clase a la que pertenece la clase anidada.**

- Es posible hacer **referencia** a la clase anidada, por ejemplo, para la declaración de variables.

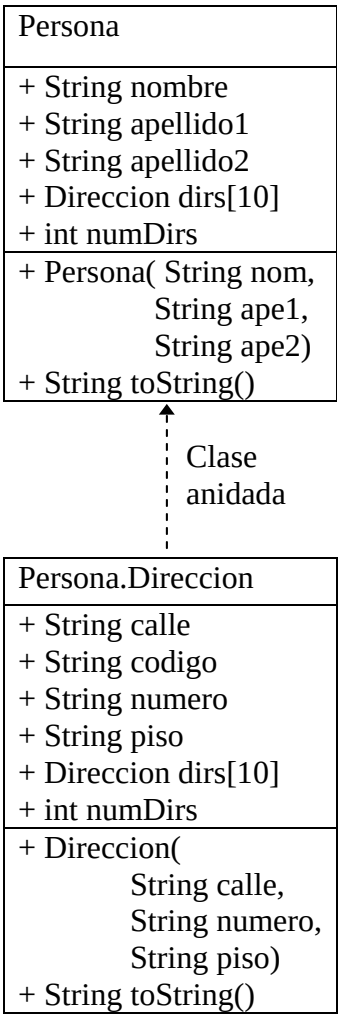
  La sintaxis es:
  `<nombre clase que anida>.<nombre clase anidada>`

- **En el código de las clases anidadas podemos utilizar atributos del objeto al que referenciamos**.

Ejemplo:

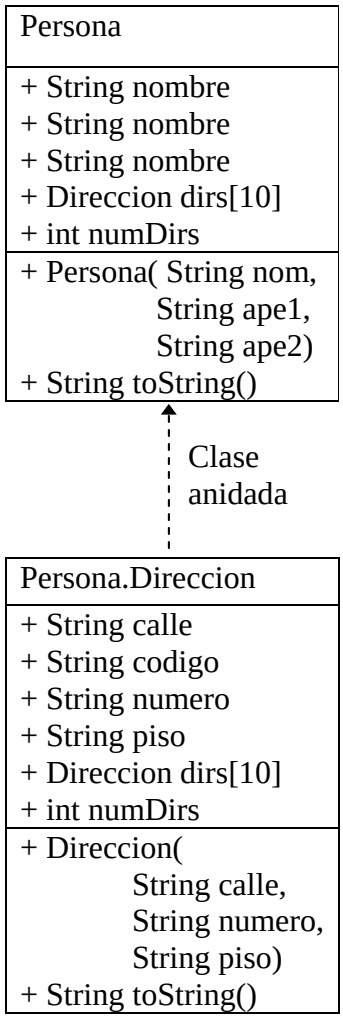
```
class A{
      int i;
      class B{
            int j;
            public B(){
                  j = i+10;
            }
      }
}
…
{
      A obja = new A();
      A.B objab = obja.new B();
}
…
```

Clases internas o anidadas (2)




```
import java.util.Scanner;

class Persona{
        public String nombre;
        public String apellido1;
        public String apellido2;
        public Direccion [] dirs = new Direccion[10];
        public int numDirs = 0;
        public Persona(   String nom,
                          String ape1,
                          String ape2 ){

                nombre = nom;
                apellido1 = ape1;
                apellido2 = ape2;
        }

        public Persona(   String nom, String ape1, String ape2,
                          Direccion dir){
                this( nom,ape1,ape2 );
                dirs[numDirs++]=dir;
        }

        /* Clase anidada */
        public class Direccion{
                public String calle;
                public String codigo;
                public int numero;
                public String piso;
                public Direccion( String nc, int nn, String np ){
                        calle = nc;
                        numero = nn;
                        piso = np;
                        codigo = new String( ""+nombre.charAt(0)
                                        +apellido1.charAt(0)
                                        +apellido2.charAt(0)
                                        +calle.charAt(0)
                                        +numero );
                }
                public String toString(){
                        return new String( codigo+":"+calle+",nº: "
                                        +numero+" piso: "+piso );
                }
        }
        /* Fin de clase anidada */

        public void aniadirDir( Direccion dir ){
                if (numDirs<10)
                        dirs[numDirs++]=dir;
        }

        public String toString(){
                return new String( ""+nombre+" "+apellido1
                                                +" "+apellido2 );
        }
}
```

Clases internas o anidadas (3)




```
public class Programa {

        public static void main(String[] args){

                Scanner teclado = new Scanner( System.in );
                System.out.println( "Introduzca nombre y apellidos" );
                Persona p = new Persona( teclado.nextLine(),
                                         teclado.nextLine(),
                                         teclado.nextLine());
                System.out.println( "¿Cuántas direcciones?" );
                for (  int i = Integer.parseInt(teclado.nextLine());
                       i>0; i-- ){
                    Persona.Direccion dir;
                    System.out.println( "Introduzca una dirección
(calle, número, piso):" );
                    dir = p.new Direccion( teclado.nextLine(),
                            Integer.parseInt(teclado.nextLine()),
                            teclado.nextLine());
                    p.aniadirDir( dir );
                }

                System.out.println( "Persona introducida:\n"+p );
                if ( p.numDirs > 0 ){
                    Persona.Direccion tmp; // Sólo por ejemplo
                    for ( int i=0; i<p.numDirs ; i++ ){
                        tmp = p.dirs[i]; // Sólo por ejemplo
                        System.out.println( "\t" + tmp );
                    }
                }else
                    System.out.println( "No se conocen
direcciones" );
        }
}
```

Object (1) (java.lang.object)

Métodos generales **de Object:**

`protected Object clone():`
Crea y devuelve una copia de este objeto. **Si se quiere implementar para clonar objetos, será necesario implementar el interfaz *Cloneable*.**

`boolean equals(Object obj):`
Verdadero cuando *obj* "es igual" a este objeto.

`protected void finalize():`
Es llamado por el recolector de basura cuando éste determina que no quedan referencias al objeto.

`Class<? extends Object> getClass():`
Devuelve la clase de un objeto en tiempo de ejecución.

`int hashCode():`
Devuelve el valor *hash* de un objeto.

`String toString():`
Devuelve una cadena representación del objeto.

Punto implements Cloneable
+ double x + double y
+ Punto(double nx, double ny) + String toString() + boolean equals(Object o) + Object clone() throws CloneNotSupportedException

En realidad, se suele implementar con una llama a **super.clone()**

```
import java.util.Scanner;
import java.util.Locale;

class Punto implements Cloneable{
        public double x;
        public double y;

        public Punto( double nx, double ny ){
                x = nx;
                y = ny;
        }

        public boolean equals( Object o ){
                if ( !(o instanceof Punto) ) return false;
                Punto otro = (Punto)o;
                return (Math.abs(otro.x - x) < 1e-10)
                        &&(Math.abs(otro.y - y) < 1e-10);
        }

        public Object clone() throws
                                CloneNotSupportedException{
                return (Object)(new Punto( x,y ));
        }
        public String toString(){
                return new String( "("+x+","+y+")" );
        }
}
```

Object (2) (java.lang.object)

```
public class Programa {

	public static void main(String[] args) throws Exception {
		// Lectura de un punto:
		Scanner s = new Scanner( System.in );
		s.useLocale( Locale.ENGLISH );
		System.out.println( "Coordenadas del primer punto" );
		Punto p1 = new Punto( s.nextDouble(),
				s.nextDouble());
		System.out.println( "Coordenadas del segundo punto" );
		Punto p2 = new Punto( s.nextDouble(),
				 s.nextDouble());
		System.out.println( p1 + (p1.equals(p2)?"==":"!=") + p2);
		Punto p3 = (Punto)p2.clone();
		System.out.println( p2 + (p2.equals(p3)?"==":"!=") + p3);

		p3.x=0;
		p3.y=0;
		System.out.println( p2 + (p2.equals(p3)?"==":"!=") + p3);

		Object obj = new Integer(4);
		System.out.println( p3 + (p3.getClass().equals(obj.getClass())?" misma":"
distinta")+" clase " + obj);
		obj = new Punto(10,10);
		System.out.println( p3 + (p3.getClass().equals(obj.getClass())?" misma":"
distinta")+" clase " + obj);
	}
}
```

String (java.lang.String) (1)

`char` **`charAt`**`(int index)`
Returns the `char` value at the specified index.
`int` **`compareTo`**`(String anotherString)`
Compares two strings lexicographically.
`int` **`compareToIgnoreCase`**`(String str)`
Compares two strings lexicographically, ignoring case differences.
`String` **`concat`**`(String str)`
Concatenates the specified string to the end of this string.
`boolean` **`contains`**`(CharSequence s)`
Returns true if and only if this string contains the specified sequence of char values.
`boolean` **`endsWith`**`(String suffix)`
Tests if this string ends with the specified suffix.
`boolean` **`equals`**`(Object anObject)`
Compares this string to the specified object.
`boolean` **`equalsIgnoreCase`**`(String anotherString)`
Compares this `String` to another `String`, ignoring case considerations.
`byte[]` **`getBytes`**`()`
Encodes this `String` into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
`int` **`hashCode`**`()`
Returns a hash code for this string.
`int` **`indexOf`**`(int ch)`
Returns the index within this string of the first occurrence of the specified character.
`int` **`indexOf`**`(int ch, int fromIndex)`
Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
`int` **`indexOf`**`(String str)`
Returns the index within this string of the first occurrence of the specified substring.
`int` **`indexOf`**`(String str, int fromIndex)`
Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
`int` **`lastIndexOf`**`(int ch)`
Returns the index within this string of the last occurrence of the specified character.
`int` **`lastIndexOf`**`(int ch, int fromIndex)`
Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
`int` **`lastIndexOf`**`(String str)`
Returns the index within this string of the rightmost occurrence of the specified substring.
`int` **`lastIndexOf`**`(String str, int fromIndex)`
Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
`int` **`length`**`()`
Returns the length of this string.

`String` **`replace`**`(char oldChar, char newChar)`
Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

`boolean` **`startsWith`**`(String prefix)`
Tests if this string starts with the specified prefix.
`boolean` **`startsWith`**`(String prefix, int toffset)`
Tests if this string starts with the specified prefix beginning a specified index.
`String` **`substring`**`(int beginIndex)`
Returns a new string that is a substring of this string.
`String` **`substring`**`(int beginIndex, int endIndex)`
Returns a new string that is a substring of this string.
`char[]` **`toCharArray`**`()`
Converts this string to a new character array.
`String` **`toLowerCase`**`()`
Converts all of the characters in this `String` to lower case using the rules of the default locale.
`String` **`toString`**`()`
This object (which is already a string!) is itself returned.
`String` **`toUpperCase`**`()`
Converts all of the characters in this `String` to upper case using the rules of the default locale.
`String` **`trim`**`()`
Returns a copy of the string, with leading and trailing whitespace omitted.
`static String` **`valueOf`**`(boolean b)`
Returns the string representation of the `boolean` argument.
`static String` **`valueOf`**`(char c)`
Returns the string representation of the `char` argument.
`static String` **`valueOf`**`(char[] data)`
Returns the string representation of the `char` array argument.
`static String` **`valueOf`**`(char[] data, int offset, int count)`
Returns the string representation of a specific subarray of the `char` array argument.
`static String` **`valueOf`**`(double d)`
Returns the string representation of the `double` argument.
`static String` **`valueOf`**`(float f)`
Returns the string representation of the `float` argument.
`static String` **`valueOf`**`(int i)`
Returns the string representation of the `int` argument.
`static String` **`valueOf`**`(long l)`
Returns the string representation of the `long` argument.
`static String` **`valueOf`**`(Object obj)`
Returns the string representation of the `Object` argument.

String (java.lang.String) (3)

```
public class Programa {

	public static void main(String[] args){
		String cadena = "Inicial";
		cadena = cadena.toUpperCase();
		System.out.println( cadena );
		cadena = cadena.substring(1,cadena.length()-1 );
		System.out.println( cadena );
		cadena = "i"+cadena+"l";
		System.out.println( cadena );
		cadena = cadena.replace( 'I','i' );
		System.out.println( cadena );
		for ( int i=1; i<cadena.length(); i+=2 )
			System.out.println( cadena.charAt(i) );

		cadena = String.valueOf( true )+" "+
				String.valueOf( 5 )+" "+
				String.valueOf( 2.12 );
		System.out.println( cadena );
	}
}
```

Salida:

```
INICIAL
NICIA
iNICIAl
iNiCiAl
N
C
A
true 5 2.12
```

StringBuffer (java.lang.StringBuffer)

La clase String almacena una cadena constante, de modo que cuando se realizan variaciones directas sobre ellas (métodos de cambio de caso, de sustitución, etc) lo que se hace es crear una cadena nueva diferente a la anterior, con el consecuente gasto de memoria.

Para evitar este efecto, ***Java*** **incluye una clase para realizar trabajos con cadenas que se van a modificar con frecuencia:** ***StringBuffer***.

Sus **métodos principales** son ***append*** e ***insert***. Ambos métodos están sobrecargados para trabajar con los diferentes tipos primitivos. Además, gracias al método ***toString***,, las clases definidas por el programador se pueden adaptar para trabajar correctamente con esta clase.

append **añade al final la cadena** que pasemos como argumento o la cadena de representación del argumento.

insert **inserta** la cadena o representación pasada como segundo argumento **en la posición especificada** en el primer argumento.

Son también útiles ***length***, ***setLength*** y ***subString***.

```
public class Programa {

        public static void main(String[] args){
                StringBuffer buffer = new StringBuffer( "Inicial" );
                buffer.append("final");
                buffer.insert(7," ");
                buffer.delete( 0,2 );
                buffer.insert( 0,"INI" );
                buffer.append(" "+buffer.length());
                String cadena = buffer.toString();
                System.out.println( cadena );
        }

}
```

Salida:

```
INIicial final 14
```

Colecciones (java.util.Collection)

- *Java* implementa una serie de interfaces y clases estándar que nos permiten almacenar objetos.
- El interfaz más general es **Collection** (**java.util.Collection**). Una clase que implementa **Collection** debe verse como una colección de objetos.

- Del interfaz **Collection** descienden otros subinterfaces, entre los que destacan: **Set**, **Queue** y **List**.

Collection<E>

`boolean` **`add`**`(E o)`
Ensures that this collection contains the specified element (optional operation).
`boolean` **`addAll`**`(Collection<? extends E> c)`
Adds all of the elements in the specified collection to this collection (optional operation).
`void` **`clear`**`()`
Removes all of the elements from this collection (optional operation).
`boolean` **`contains`**`(Object o)`
Returns `true` if this collection contains the specified element.
`boolean` **`containsAll`**`(Collection<?> c)`
Returns `true` if this collection contains all of the elements in the specified collection.
`boolean` **`equals`**`(Object o)`
Compares the specified object with this collection for equality.
`int` **`hashCode`**`()`
Returns the hash code value for this collection.
`boolean` **`isEmpty`**`()`
Returns `true` if this collection contains no elements.
`Iterator<E>` **`iterator`**`()`
Returns an iterator over the elements in this collection.
`boolean` **`remove`**`(Object o)`
Removes a single instance of the specified element from this collection, if it is present (optional operation).
`boolean` **`removeAll`**`(Collection<?> c)`
Removes all this collection's elements that are also contained in the specified collection (optional operation).
`boolean` **`retainAll`**`(Collection<?> c)`
Retains only the elements in this collection that are contained in the specified collection (optional operation).
`int` **`size`**`()`
Returns the number of elements in this collection.
`Object[]` **`toArray`**`()`
Returns an array containing all of the elements in this collection.
`<T> T[]` **`toArray`**`(T[] a)`
Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

Colecciones (java.util.List)

- *Java* implementa un interfaz genérico para listas. Básicamente, una lista es una colección que puede contener objetos repetidos y en la que los objetos están ordenados en base a un índice.

interface **List<E>** (java.util)

`boolean` **`add`**`(E o)`
Appends the specified element to the end of this list (optional operation).

`void` **`add`**`(int index, E element)`
Inserts the specified element at the specified position in this list (optional operation).

`boolean` **`addAll`**`(Collection<? extends E> c)`
Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).

`boolean` **`addAll`**`(int index, Collection<? extends E> c)`
Inserts all of the elements in the specified collection into this list at the specified position (optional operation).

`void` **`clear`**`()`
Removes all of the elements from this list (optional operation).

`boolean` **`contains`**`(Object o)`
Returns `true` if this list contains the specified element.

`boolean` **`containsAll`**`(Collection<?> c)`
Returns `true` if this list contains all of the elements of the specified collection.

`boolean` **`equals`**`(Object o)`
Compares the specified object with this list for equality.

`E` **`get`**`(int index)`
Returns the element at the specified position in this list.

`int` **`hashCode`**`()`
Returns the hash code value for this list.

`int` **`indexOf`**`(Object o)`
Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.

`boolean` **`isEmpty`**`()`
Returns `true` if this list contains no elements.

`Iterator<E>` **`iterator`**`()`
Returns an iterator over the elements in this list in proper sequence.

`int` **`lastIndexOf`**`(Object o)`
Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.

`ListIterator<E>` **`listIterator`**`()`
Returns a list iterator of the elements in this list (in proper sequence).

`ListIterator<E>` **`listIterator`**`(int index)`
Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.

`E` **`remove`**`(int index)`
Removes the element at the specified position in this list (optional operation).

`boolean` **`remove`**`(Object o)`
Removes the first occurrence in this list of the specified element (optional operation).

`boolean` **`removeAll`**`(Collection<?> c)`
Removes from this list all the elements that are contained in the specified collection (optional operation).

Colecciones (java.util.List)

`boolean` **`retainAll`**`(Collection<?> c)`
Retains only the elements in this list that are contained in the specified collection (optional operation).

`E` **`set`**`(int index, E element)`
Replaces the element at the specified position in this list with the specified element (optional operation).

`int` **`size`**`()`
Returns the number of elements in this list.

`List<E>` **`subList`**`(int fromIndex, int toIndex)`
Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.

`Object[]` **`toArray`**`()`
Returns an array containing all of the elements in this list in proper sequence.

`<T> T[]` **`toArray`**`(T[] a)`
Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.

Colecciones (java.util.ArrayList)

ArrayList<E>

- *Java* implementa la clase **ArrayList** (**ArrayList<E>** a partir de **Java 5**) que implementa una lista.

```
import java.util.ArrayList;

class Punto1D{
	public double x;
	public Punto1D( double nx ){
		x = nx;
	}
	public String toString(){
		return new String( "("+x+")" );
	}
}

public class Programa {

	public static void main(String[] args) throws Exception {
		ArrayList<Punto1D> arrayPunto = new ArrayList<Punto1D>();
		// añadir 10 elementos...
		for ( int i=0; i<10; i++ ){
			arrayPunto.add( new Punto1D(i) );
		}
		System.out.println( arrayPunto );
		// eliminar los elementos en las posiciones pares
		for ( int i=arrayPunto.size()-1; i>=0 ; i-- ){
			if ( i%2 == 0 )
				arrayPunto.remove( i );
		}
		System.out.println( arrayPunto );
		// creamos una lista de objetos con contenido inicial
		// igual a la lista de puntos que llevamos
		ArrayList arrayObjeto = new ArrayList( arrayPunto );
		// Cambiamos los objetos que haya en las posiciones impares
		// por cadenas "#"
		for ( int i=arrayObjeto.size()-1; i>=0 ; i-- ){
			if ( i%2 == 1 ){
				arrayObjeto.remove( i );
				arrayObjeto.add( i,new String("#") );
			}
		}
		System.out.println( arrayObjeto );
	}
}
```

- La clase **ArrayList<E>** implementa los interfaces **Collection** y **List**.
- Existe también la clase **Vector** (java.util) que implementa también una lista. **Vector** está implementada de modo que se puede utilizar concurrentemente y, por ello, es más lenta que **ArrayList**. Por otra parte, **ArrayList** se puede utilizar de forma segura para el acceso concurrente utilizando las características preprogramadas que provee la API estándar de *Java*. Se recomienda la utilización de **ArrayList** en lugar de **Vector**.

Iteradores (java.lang.Iterable, java.util.Iterator)

- Un iterador es una clase que nos permite recorrer una estructura que almacena datos.
- Para crear un iterador, se crea una clase que implementa el interfaz **Iterator** (**Iterator<E>** a partir de **Java 5**).
- Los métodos de **Iterator<E>** son:

class **Iterator<E>** (java.util)

`boolean` **`hasNext`**`()`
Returns `true` if the iteration has more elements.
`E` **`next`**`()`
Returns the next element in the iteration.
`void` **`remove`**`()`
Removes from the underlying collection the last element returned by the iterator (optional operation).

- Si queremos que nuestra clase pueda ser recorrida por un iterador, lo que haremos es declararla iterable. Esto se hace a través del interfaz **Iterable** (java.lang)
- **Iterable** es superinterfaz de **Collection**. Esto quiere decir que una colección debe ser iterable (se puede obtener a partir de ella un iterador que nos permita recorrer todos los valores que contiene la colección).
- El único método de **Iterable** es:

interface **Iterable** (java.lang)

`Iterator<T>` **`iterator`**`()`
Returns an iterator over a set of elements of type T.

Y que devuelve un iterador de elementos de clase T.

Ejercicio propuesto:

Programar una clase **ArbolBin<E>** que sea una colección que guarda los elementos en una estructura de árbol binario. Haremos, además, una clase **ArbolBinIter<E>**, que implemente una clase iterador para el árbol binario programado.

Realizar un programa de prueba utilizando la sintaxis del bucle **for** incluida en *Java 5* (Transparencia siguiente)

Bucle for mejorado (Java 5)

- En *Java 5* se he extendido la sintaxis del bucle **for** para trabajar con colecciones de forma más natural. En realidad, esta extensión nos permite trabajar con iterables (**java.lang.Iterable**), que, a su vez, es superinterfaz de **Collection**.

  La sintaxis es:

```
for ( <clase> <id> : <iterable> ){
      …
}
```

  Ejemplo:

```
ArrayList array = new ArrayList();
…
// muestra todos los objetos en array
for ( Object elemento : array )
      System.out.println( elemento );
```

  Funciona también con arrays normales ()

```
String [] array = new String[10];
…
// muestra todas las cadenas en array
for ( String st : array )
      System.out.println( st );
```

```
import java.util.Scanner;

public class Programa {

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

                final int TOTAL_NUMEROS = 10;

                Scanner teclado = new Scanner( System.in );
                Integer [] array = new Integer[TOTAL_NUMEROS];

                System.out.println( "Introduzca "+TOTAL_NUMEROS+" enteros." );

                for ( int i=0; i<TOTAL_NUMEROS; i++ ){
                        array[i] = new Integer(Integer.parseInt(teclado.nextLine()));
                }

                System.out.println( "Se introdujeron:" );

                for ( Integer numero : array )
                        System.out.println( numero );

        }
}
```

Random (java.util).

- La clase **Random** (java.util) implementa un generador de números aleatorios. Sus métodos son:

`protected int` **`next`**`(int bits)`
Generates the next pseudorandom number.

`boolean` **`nextBoolean`**`()`
Returns the next pseudorandom, uniformly distributed `boolean` value from this random number generator's sequence.

`void` **`nextBytes`**`(byte[] bytes)`
Generates random bytes and places them into a user-supplied byte array.

`double` **`nextDouble`**`()`
Returns the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence.

`float` **`nextFloat`**`()`
Returns the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence.

`double` **`nextGaussian`**`()`
Returns the next pseudorandom, Gaussian ("normally") distributed `double` value with mean `0.0` and standard deviation `1.0` from this random number generator's sequence.

`int` **`nextInt`**`()`
Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.

`int` **`nextInt`**`(int n)`
Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

`long` **`nextLong`**`()`
Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence.

`void` **`setSeed`**`(long seed)`
Sets the seed of this random number generator using a single `long` seed.

Math (java.lang)

- La clase **Math** es una clase no instanciable que contiene las definiciones de algunas constantes matemáticas (e, pi) y funciones matemáticas. Sus definiciones son:

class **Math** (java.lang)

Constantes:

`static double` **`E`**
The `double` value that is closer than any other to *e*, the base of the natural logarithms.
`static double` **`PI`**
The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Métodos:

`static double` **`abs`**`(double a)`
Returns the absolute value of a `double` value.
`static float` **`abs`**`(float a)`
Returns the absolute value of a `float` value.
`static int` **`abs`**`(int a)`
Returns the absolute value of an `int` value.
`static long` **`abs`**`(long a)`
Returns the absolute value of a `long` value.
`static double` **`acos`**`(double a)`
Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
`static double` **`asin`**`(double a)`
Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.
`static double` **`atan`**`(double a )`
Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.
`static double` **`atan2`**`(double y, double x)`
Converts rectangular coordinates (`x`, `y`) to polar (r, *theta*).
`static double` **`cbrt`**`(double a)`
Returns the cube root of a `double` value.
`static double` **`ceil`**`(double a)`
Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer.
`static double` **`cos`**`(double a)`
Returns the trigonometric cosine of an angle.
`static double` **`cosh`**`(double x)`
Returns the hyperbolic cosine of a `double` value.
`static double` **`exp`**`(double a)`
Returns Euler's number *e* raised to the power of a `double` value.
`static double static double` **`hypot`**`(double x, double y)`
Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.
`static double static double` **`log`**`(double a)`
Returns the natural logarithm (base *e*) of a `double` value.

`static double` **`log10`**`(double a)`
Returns the base 10 logarithm of a `double` value.
`static double` **`max`**`(double a, double b)`
Returns the greater of two `double` values.
`static float` **`max`**`(float a, float b)`
Returns the greater of two `float` values.
`static int` **`max`**`(int a, int b)`
Returns the greater of two `int` values.
`static long` **`max`**`(long a, long b)`
Returns the greater of two `long` values.
`static double` **`min`**`(double a, double b)`
Returns the smaller of two `double` values.
`static float` **`min`**`(float a, float b)`
Returns the smaller of two `float` values.
`static int` **`min`**`(int a, int b)`
Returns the smaller of two `int` values.
`static long` **`min`**`(long a, long b)`
Returns the smaller of two `long` values.
`static double` **`pow`**`(double a, double b)`
Returns the value of the first argument raised to the power of the second argument.
`static double` **`random`**`()`
Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static double` **`rint`**`(double a)`
Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.
`static long` **`round`**`(double a)`
Returns the closest `long` to the argument.
`static int` **`round`**`(float a)`
Returns the closest `int` to the argument.
`static double` **`signum`**`(double d)`
Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
`static float` **`signum`**`(float f)`
Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
`static double` **`sin`**`(double a)`
Returns the trigonometric sine of an angle.
`static double` **`sqrt`**`(double a)`
Returns the correctly rounded positive square root of a `double` value.
`static double` **`tan`**`(double a)`
Returns the trigonometric tangent of an angle.
`static double` **`toDegrees`**`(double angrad)`
Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
`static double` **`toRadians`**`(double angdeg)`
Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

Ejemplo con Math y Random.

Aproximación iterativa de PI.

Utilizando la fórmula de la superficie de un círculo (PI*radio^2) y un generador de números aleatorios, aproximar el valor de PI.

Solución:

- Un círculo de radio 1 tiene una superficie igual a PI
- Un círculo de radio 1 se puede inscribir en un cuadrado de lado 2
- Sabemos que un punto está dentro del círculo de radio 1 si su distancia al centro es >=1
- La distancia se puede calcular con el teorema de Pitágoras
- La superficie de un cuadrado de lado 2 es 2*2 = 4
- Si nosotros generamos puntos al azar (distribuidos de forma uniforme) dentro de un cuadrado de lado 2 y contamos los que caen dentro del círculo y los totales, podemos aproximar la superficie que ocupa en círculo (que, en este caso, es PI)
    - El total de puntos generados corresponderá a la superficie total del cuadrado.
    - El total de puntos que hayan caído dentro del círculo dividido entre el número total de puntos generados me dará la proporción de la superficie que cae dentro del círculo.
    - La superficie del círculo de radio 1 será la proporción de puntos que cayeron dentro de él multiplicada por la superficie total.

```
import java.util.Random;

public class Programa {

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

		final long TOTAL_PUNTOS = 1000000;
		long puntosDentro=0;
		Random generador = new Random();
		double x;
		double y;

		for ( int i=0; i<TOTAL_PUNTOS; i++ ){

			//generamos el punto en el cuadrado con esquinas (-1,-1) y (1,1)

			x = (generador.nextDouble()-0.5)*2;
			y = (generador.nextDouble()-0.5)*2;

			// Si cae dentro del círculo, incrementamos la cuenta
			if ( Math.hypot(x,y) <= 1 )
				puntosDentro++;
		}
		/* La superficie total es 4 y la del círculo será la proporción de puntos que
		 * cayeron dentro del círculo con respecto al total de puntos generados
		 */
		System.out.println( "La aproximación de PI con "+ TOTAL_PUNTOS +
			" puntos eleatorios es: "+((double)puntosDentro/TOTAL_PUNTOS)*4 );
	}
}
```

Flujos y archivos

Clase File

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca la ruta" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		if ( archivo.exists() ){

			if ( archivo.isDirectory() ){
				System.out.println( archivo.getAbsolutePath()
						+" es un directorio con contenido:" );
				File [] listado = archivo.listFiles();
				for ( File f : listado ){
					System.out.println( f.getAbsolutePath() );
				}
			}else{
				System.out.println( archivo.getAbsolutePath()
						+" es un archivo");
			}
		}else{
			System.out.println( "No se encuentra la ruta especificada" );
		}
	}
}
```

Flujos con caracteres (modo texto)

Writer
- **BufferedWriter**
- **CharArrayWriter**
- **FilterWriter**
- **OutputStreamWriter**
  - **FileWriter**
- **PrintWriter**
- **StringWriter**

Reader
- **BufferedReader**
  - **LineNumberReader**
- **CharArrayReader**
- **FilterReader**
- **InputStreamReader**
  - **FileReader**
- **StringReader**

En la lectura, sabremos que hemos llegado al final del archivo cuando se produzca una excepción **EOFException** al intentar una lectura:

```
try{
     while( true ){
          cadena = archivo.read();
          …
     }
}catch( EOFException e ){}
```

Clase FileWriter:

Constructor Summary

FileWriter(File file) Constructs a FileWriter object given a File object.
FileWriter(File file, boolean append) Constructs a FileWriter object given a File object.
FileWriter(FileDescriptor fd) Constructs a FileWriter object associated with a file descriptor.
FileWriter(String fileName) Constructs a FileWriter object given a file name.
FileWriter(String fileName, boolean append) Constructs a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

Method Summary

Methods inherited from class java.io.OutputStreamWriter
close, flush, getEncoding, write, write, write

Methods inherited from class java.io.Writer
append, append, append, write, write

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo .txt" );
		String ruta = teclado.nextLine();
		try{
			FileWriter escritura = new FileWriter( ruta );
			escritura.write( "Cadena que se escribe en el archivo");
			escritura.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Clase FileReader

Constructor Summary

FileReader(File file) Creates a new FileReader, given the File to read from.
FileReader(FileDescriptor fd) Creates a new FileReader, given the FileDescriptor to read from.
FileReader(String fileName) Creates a new FileReader, given the name of the file to read from.

Method Summary

Methods inherited from class java.io.InputStreamReader
close, getEncoding, read, read, ready

Methods inherited from class java.io.Reader
mark, markSupported, read, read, reset, skip

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo .txt" );
		String ruta = teclado.nextLine();
		try{
			char [] cadena = new char[100];
			FileReader lectura = new FileReader( ruta );
			// leemos hasta que se llene el array o termine el archivo
			lectura.read( cadena );
			lectura.close();
			System.out.println( (new String(cadena)).trim() );
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

BufferedWriter, BufferedReader y encadenamiento de salidas/entradas

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo .txt" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			// En este ejemplo, se abre el archivo en modo de añadir
			FileWriter escritura = new FileWriter( archivo, true );
			BufferedWriter buffer = new BufferedWriter( escritura );
			buffer.write("Cadena que se guarda\n");
			buffer.close();
			escritura.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo .txt" );
		String ruta = teclado.nextLine();
		try{
			FileReader lectura = new FileReader( ruta );
			BufferedReader buffer = new BufferedReader( lectura );
			// leemos una cadena
			String cadena = buffer.readLine();
			buffer.close();
			lectura.close();
			System.out.println( cadena );
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Flujos con bytes (modo binario)

InputStream
- **ByteArrayInputStream**
- **FileInputStream**
- **FilterInputStream**
  - **BufferedInputStream**
  - **DataInputStream**
- **ObjectInputStream**
- **StringBufferInputStream**

OutputStream
- **ByteArrayOutputStream**
- **FileOutputStream**
- **FilterOutputStream**
  - **BufferedOutputStream**
  - **DataOutputStream**
  - **PrintStream**
- **ObjectOutputStream**

En la lectura, sabremos que hemos llegado al final del archivo cuando se produzca una excepción **EOFException** al intentar una lectura:

```
try{
     while( true ){
          dato = archivo.read();
          …
     }
}catch( EOFException e ){}
```

Clase FileOutputStream

Constructor Summary

FileOutputStream(File file) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(File file, boolean append) Creates a file output stream to write to the file represented by the specified File object.
FileOutputStream(FileDescriptor fdObj) Creates an output file stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.
FileOutputStream(String name) Creates an output file stream to write to the file with the specified name.
FileOutputStream(String name, boolean append) Creates an output file stream to write to the file with the specified name.

Method Summary

void	**close**() Closes this file output stream and releases any system resources associated with this stream.
protected void	**finalize**() Cleans up the connection to the file, and ensures that the close method of this file output stream is called when there are no more references to this stream.
FileChannel	**getChannel**() Returns the unique FileChannel object associated with this file output stream.
FileDescriptor	**getFD**() Returns the file descriptor associated with this stream.
void	**write**(byte[] b) Writes b.length bytes from the specified byte array to this file output stream.
void	**write**(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to this file output stream.
void	**write**(int b) Writes the specified byte to this file output stream.

Methods inherited from class java.io.OutputStream
flush

Clase FileInputStream

Constructor Summary

FileInputStream(File file) Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.
FileInputStream(FileDescriptor fdObj) Creates a FileInputStream by using the file descriptor fdObj, which represents an existing connection to an actual file in the file system.
FileInputStream(String name) Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system.

Method Summary

int	**available**() Returns the number of bytes that can be read from this file input stream without blocking.
void	**close**() Closes this file input stream and releases any system resources associated with the stream.
protected void	**finalize**() Ensures that the close method of this file input stream is called when there are no more references to it.
FileChannel	**getChannel**() Returns the unique FileChannel object associated with this file input stream.
FileDescriptor	**getFD**() Returns the FileDescriptor object that represents the connection to the actual file in the file system being used by this FileInputStream.
int	**read**() Reads a byte of data from this input stream.
int	**read**(byte[] b) Reads up to b.length bytes of data from this input stream into an array of bytes.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from this input stream into an array of bytes.
long	**skip**(long n) Skips over and discards n bytes of data from the input stream.

Methods inherited from class java.io.InputStream
mark, markSupported, reset

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			FileOutputStream escritura =new FileOutputStream(
archivo );
			byte [] array = new byte[5];
			for ( byte i=0; i<5; i++ )
				array[i] = i;
			escritura.write( array );
			escritura.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			FileInputStream lectura =new FileInputStream( archivo );
			byte [] array = new byte[5];
			lectura.read( array );
			lectura.close();
			for ( int i=0; i<5; i++ )
				System.out.println( array[i] );
			}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

clase DataOutputStream

Constructor Summary
DataOutputStream(OutputStream out) Creates a new data output stream to write data to the specified underlying output stream.

Method Summary	
void	**flush**() Flushes this data output stream.
int	**size**() Returns the current value of the counter written, the number of bytes written to this data output stream so far.
void	**write**(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to the underlying output stream.
void	**write**(int b) Writes the specified byte (the low eight bits of the argument b) to the underlying output stream.
void	**writeBoolean**(boolean v) Writes a boolean to the underlying output stream as a 1-byte value.
void	**writeByte**(int v) Writes out a byte to the underlying output stream as a 1-byte value.
void	**writeBytes**(String s) Writes out the string to the underlying output stream as a sequence of bytes.
void	**writeChar**(int v) Writes a char to the underlying output stream as a 2-byte value, high byte first.
void	**writeChars**(String s) Writes a string to the underlying output stream as a sequence of characters.
void	**writeDouble**(double v) Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first.
void	**writeFloat**(float v) Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.
void	**writeInt**(int v) Writes an int to the underlying output stream as four bytes, high byte first.
void	**writeLong**(long v) Writes a long to the underlying output stream as eight bytes, high byte first.
void	**writeShort**(int v) Writes a short to the underlying output stream as two bytes, high byte first.
void	**writeUTF**(String str) Writes a string to the underlying output stream using modified UTF-8 encoding in a machine-independent manner.

Methods inherited from class java.io.FilterOutputStream
close, write

Methods inherited from interface java.io.DataOutput
write

clase DataInputStream

Constructor Summary
DataInputStream(InputStream in) Creates a DataInputStream that uses the specified underlying InputStream.

Method Summary	
int	**read**(byte[] b) Reads some number of bytes from the contained input stream and stores them into the buffer array b.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from the contained input stream into an array of bytes.
boolean	**readBoolean**() See the general contract of the readBoolean method of DataInput.
byte	**readByte**() See the general contract of the readByte method of DataInput.
char	**readChar**() See the general contract of the readChar method of DataInput.
double	**readDouble**() See the general contract of the readDouble method of DataInput.
float	**readFloat**() See the general contract of the readFloat method of DataInput.
void	**readFully**(byte[] b) See the general contract of the readFully method of DataInput.
void	**readFully**(byte[] b, int off, int len) See the general contract of the readFully method of DataInput.
int	**readInt**() See the general contract of the readInt method of DataInput.
long	**readLong**() See the general contract of the readLong method of DataInput.
short	**readShort**() See the general contract of the readShort method of DataInput.
int	**readUnsignedByte**() See the general contract of the readUnsignedByte method of DataInput.
int	**readUnsignedShort**() See the general contract of the readUnsignedShort method of DataInput.
String	**readUTF**() See the general contract of the readUTF method of DataInput.
static String	**readUTF**(DataInput in) Reads from the stream in a representation of a Unicode character string encoded in modified UTF-8 format; this string of characters is then returned as a String.
int	**skipBytes**(int n) See the general contract of the skipBytes method of DataInput.

Methods inherited from class java.io.FilterInputStream
available, close, mark, markSupported, read, reset, skip

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
		    FileOutputStream escritura=new FileOutputStream(archivo);
		    DataOutputStream datos=new DataOutputStream(escritura);
		    for ( int i=0; i<5;i++ )
		        datos.writeInt( i );
		    datos.close();
		    escritura.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			FileInputStream lectura =new FileInputStream( archivo );
			DataInputStream datos =new DataInputStream( lectura );
			try{
				while ( true ){
					System.out.println( datos.readInt() );
				}
			}catch( EOFException e ){}
			datos.close();
			lectura.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Archivos de acceso aleatorio

Clase RandomAccessFile

Constructor Summary
RandomAccessFile(File file, String mode) Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode) Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Method Summary	
void	**close**() Closes this random access file stream and releases any system resources associated with the stream.
FileChannel	**getChannel**() Returns the unique FileChannel object associated with this file.
FileDescriptor	**getFD**() Returns the opaque file descriptor object associated with this stream.
long	**getFilePointer**() Returns the current offset in this file.
long	**length**() Returns the length of this file.
int	**read**() Reads a byte of data from this file.
int	**read**(byte[] b) Reads up to b.length bytes of data from this file into an array of bytes.
int	**read**(byte[] b, int off, int len) Reads up to len bytes of data from this file into an array of bytes.
boolean	**readBoolean**() Reads a boolean from this file.
byte	**readByte**() Reads a signed eight-bit value from this file.
char	**readChar**() Reads a Unicode character from this file.
double	**readDouble**() Reads a double from this file.
float	**readFloat**() Reads a float from this file.
void	**readFully**(byte[] b) Reads b.length bytes from this file into the byte array, starting at the current file pointer.
void	**readFully**(byte[] b, int off, int len) Reads exactly len bytes from this file into the byte array, starting at the current file pointer.
int	**readInt**() Reads a signed 32-bit integer from this file.
String	**readLine**() Reads the next line of text from this file.
long	**readLong**() Reads a signed 64-bit integer from this file.
short	**readShort**()

	Reads a signed 16-bit number from this file.
`int`	**`readUnsignedByte`**`()` Reads an unsigned eight-bit number from this file.
`int`	**`readUnsignedShort`**`()` Reads an unsigned 16-bit number from this file.
`String`	**`readUTF`**`()` Reads in a string from this file.
`void`	**`seek`**`(long pos)` Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
`void`	**`setLength`**`(long newLength)` Sets the length of this file.
`int`	**`skipBytes`**`(int n)` Attempts to skip over `n` bytes of input discarding the skipped bytes.
`void`	**`write`**`(byte[] b)` Writes `b.length` bytes from the specified byte array to this file, starting at the current file pointer.
`void`	**`write`**`(byte[] b, int off, int len)` Writes `len` bytes from the specified byte array starting at offset `off` to this file.
`void`	**`write`**`(int b)` Writes the specified byte to this file.
`void`	**`writeBoolean`**`(boolean v)` Writes a `boolean` to the file as a one-byte value.
`void`	**`writeByte`**`(int v)` Writes a `byte` to the file as a one-byte value.
`void`	**`writeBytes`**`(String s)` Writes the string to the file as a sequence of bytes.
`void`	**`writeChar`**`(int v)` Writes a `char` to the file as a two-byte value, high byte first.
`void`	**`writeChars`**`(String s)` Writes a string to the file as a sequence of characters.
`void`	**`writeDouble`**`(double v)` Converts the double argument to a `long` using the `doubleToLongBits` method in class `Double`, and then writes that `long` value to the file as an eight-byte quantity, high byte first.
`void`	**`writeFloat`**`(float v)` Converts the float argument to an `int` using the `floatToIntBits` method in class `Float`, and then writes that `int` value to the file as a four-byte quantity, high byte first.
`void`	**`writeInt`**`(int v)` Writes an `int` to the file as four bytes, high byte first.
`void`	**`writeLong`**`(long v)` Writes a `long` to the file as eight bytes, high byte first.
`void`	**`writeShort`**`(int v)` Writes a `short` to the file as two bytes, high byte first.
`void`	**`writeUTF`**`(String str)` Writes a string to the file using modified UTF-8 encoding in a machine-independent manner.

Modos de apertura:

Value	Meaning
`"r"`	Open for reading only. Invoking any of the `write` methods of the resulting object will cause an `IOException` to be thrown.
`"rw"`	Open for reading and writing. If the file does not already exist then an attempt will be made to create it.
`"rws"`	Open for reading and writing, as with `"rw"`, and also require that every update to the file's content or metadata be written synchronously to the underlying storage device.
`"rwd"`	Open for reading and writing, as with `"rw"`, and also require that every update to the file's content be written synchronously to the underlying storage device.

Métodos: close, seek, getFilePointer, length, setLength, write…, read…

Para el modo texto:

writeBytes, readLine

NOTA: ReadLine devuelve **null** cuando no se ha leído una cadena (p. ej. **Cuando se ha llegado al final del archivo**)

Con los archivos de acceso aleatorio, podemos saber si hemos llagado al final del archivo si el puntero está colocado en la posición *length*

```
RandomAccessFile aleatorio = new RandomAccessFile(nombre,"rw");
long tamanio = aleatorio.length();
…
while ( aleatorio.getFilePointer() < tamanio ){
      …
}
```

Modo texto

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo de texto" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			// modo escritura
			RandomAccessFile aleatorio = new RandomAccessFile(
archivo, "rw" );
			// eliminar lo que hubiera
			aleatorio.setLength(0);
			// para guardar texto en ASCII debemos guardar bytes
			aleatorio.writeBytes( "Guardar este texto\nOtra línea" );
			aleatorio.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		int numlineas = 0;
		try{
			// modo lectura
		    RandomAccessFile aleatorio = new RandomAccessFile( archivo, "r" );
			String cadena;
			while ( (cadena = aleatorio.readLine())!=null ){
				numlineas ++;
				System.out.println( cadena );
			}
			aleatorio.close();
		}catch( IOException e ){
			System.out.println( e );
		}
		System.out.println( "Total de líneas: "+numlineas );
	}
}
```

Modo binario

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			//modo escritura
		      RandomAccessFile aleatorio = new RandomAccessFile( archivo, "rw" );
			// eliminar lo que hubiera
			aleatorio.setLength(0);
			for ( int i=0; i<100; i++ )
				aleatorio.writeInt(i);
			aleatorio.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

```
import java.util.Scanner;
import java.io.*;

public class Programa {

	public static void main(String[] args){

		final int TAMANIO_ENTERO = 4;
		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			// modo lectura
			RandomAccessFile aleatorio = new RandomAccessFile( archivo, "r" );
			System.out.println( "Tamaño: " + aleatorio.length() );
			// nos posicionamos en el 57ésimo entero: (entero = 4 bytes)
			aleatorio.seek( 56*TAMANIO_ENTERO );
			System.out.println( aleatorio.readInt() );
			// nos posicionamos en el anterior: hay que multiplicar por
			// 2 porque en realidad retrocedemos 2 posiciones (cuando se
			// lee, se avanza una posición)
			aleatorio.seek( aleatorio.getFilePointer()-(2*TAMANIO_ENTERO) );
			System.out.println( aleatorio.readInt() );
			aleatorio.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Almacenamiento y recuperación de objetos en flujos: ObjectOutputStream, ObjectInputStream, Serializable, writeObject, readObject.

Salida con Serializable y writeObject con la implementación por defecto

```
import java.util.Scanner;
import java.io.*;

class Punto1D implements Serializable{

	public double x;
	public String nombre;

	public Punto1D( String nnombre, double nx ){
		nombre = nnombre;
		x = nx;
	}

	public String toString(){
		return new String( nombre+" ("+x+")" );
	}
}

public class Programa {

	public static void main(String[] args){

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			// modo lectura
			FileOutputStream salidaArchivo = new FileOutputStream( archivo );
		    ObjectOutputStream salidaObjetos = new ObjectOutputStream( salidaArchivo );
			for( int i=0; i<20; i++ ){
				Punto1D p = new Punto1D( ""+i, i*0.5 );
				salidaObjetos.writeObject( p );
			}
			salidaObjetos.close();
			salidaArchivo.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Almacenamiento y recuperación de objetos en flujos: ObjectOutputStream, ObjectInputStream, Serializable, writeObject, readObject.

Entrada con Serializable y readObject con la implementación por defecto

```
import java.util.Scanner;
import java.io.*;

class Punto1D implements Serializable{

	public double x;
	public String nombre;

	public Punto1D( String nnombre, double nx ){
		nombre = nnombre;
		x = nx;
	}

	public String toString(){
		return new String( nombre+" ("+x+")" );
	}
}

public class Programa {

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

		Scanner teclado = new Scanner( System.in );
		System.out.println( "Introduzca el nombre del archivo binario" );
		String ruta = teclado.nextLine();
		File archivo = new File( ruta );
		try{
			// modo lectura
			FileInputStream entradaArchivo = new FileInputStream( archivo );
		ObjectInputStream entradaObjetos = new ObjectInputStream( entradaArchivo );
			try{
				while (true){
					Punto1D p = (Punto1D)entradaObjetos.readObject();
					System.out.println( p );
				}
			}catch( EOFException e ){}
			entradaObjetos.close();
			entradaArchivo.close();
		}catch( IOException e ){
			System.out.println( e );
		}
	}
}
```

Almacenamiento y recuperación de objetos en flujos: ObjectOutputStream, ObjectInputStream, Serializable, writeObject, readObject.

Salida con Serializable y writeObject con implementación propia

```
import java.util.Scanner;
import java.io.*;

class Punto1D implements Serializable{

        public double x;
        public String nombre;

        public Punto1D( String nnombre, double nx ){
                nombre = nnombre;
                x = nx;
        }

        public String toString(){
                return new String( nombre+" ("+x+")" );
        }

        private void writeObject( ObjectOutputStream stream )
   throws IOException{
                // Guardamos sólo la coordenada:
                stream.writeDouble( x );
        }

         private void readObject( ObjectInputStream stream )
    throws IOException, ClassNotFoundException{
                 x = stream.readDouble();
                 nombre = new String( "leido:"+x );
         }
}

public class Programa {

        public static void main(String[] args){

                Scanner teclado = new Scanner( System.in );
                System.out.println( "Introduzca el nombre del archivo binario" );
                String ruta = teclado.nextLine();
                File archivo = new File( ruta );
                try{
                        // modo lectura
                        FileOutputStream salidaArchivo = new FileOutputStream( archivo );
                ObjectOutputStream salidaObjetos = new ObjectOutputStream( salidaArchivo );
                        for( int i=0; i<20; i++ ){
                                Punto1D p = new Punto1D( ""+i, i*0.5 );
                                salidaObjetos.writeObject( p );
                        }
                        salidaObjetos.close();
                        salidaArchivo.close();
                }catch( IOException e ){
                        System.out.println( e );
                }
        }
}
```

Almacenamiento y recuperación de objetos en flujos: ObjectOutputStream, ObjectInputStream, Serializable, writeObject, readObject.

Entrada con Serializable y readObject con implementación propia

```
import java.util.Scanner;
import java.io.*;

class Punto1D implements Serializable{

        public double x;
        public String nombre;

        public Punto1D( String nnombre, double nx ){
                nombre = nnombre;
                x = nx;
        }

        public String toString(){
                return new String( nombre+" ("+x+")" );
        }

        private void writeObject( ObjectOutputStream stream )
  throws IOException{
                // Guardamos sólo la coordenada:
                stream.writeDouble( x );
        }

         private void readObject( ObjectInputStream stream )
   throws IOException, ClassNotFoundException{
                 x = stream.readDouble();
                 nombre = new String( "leido:"+x );
         }
}

public class Programa {

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

                Scanner teclado = new Scanner( System.in );
                System.out.println( "Introduzca el nombre del archivo binario" );
                String ruta = teclado.nextLine();
                File archivo = new File( ruta );
                try{
                        // modo lectura
                        FileInputStream entradaArchivo = new FileInputStream( archivo );
                    ObjectInputStream entradaObjetos = new ObjectInputStream( entradaArchivo );
                        try{
                                while (true){
                                        Punto1D p = (Punto1D)entradaObjetos.readObject();
                                        System.out.println( p );
                                }
                        }catch( EOFException e ){}
                        entradaObjetos.close();
                        entradaArchivo.close();
                }catch( IOException e ){
                        System.out.println( e );
                }
        }
}
```