```java
/** Añadir palabra */
public static boolean add(String palabra) {
	return palabras.add(palabra.toUpperCase());
}
/** Listar palabra */
public static void print() {
	for (String palabra : palabras) {
		System.out.println(palabra);
	}
}
/** Buscar palabra */
public static boolean get(String palabra) {

	for (String s : palabras) {
		if (s.equalsIgnoreCase(palabra)) {
			return true;
		}
	}

	return false;
}
/** @return La palabra de la lista con el índice pasado por parámetro. */
public static String getPalabraConIndex(int index) {
	Iterator<String> palabrasIte = palabras.iterator();

	if (index > palabras.size()) return null;

	for (int i = 0; i ≤ index; i++) {
		if (i == index) {
			return palabrasIte.next();
		}
		palabrasIte.next();
	}

	return null;
}
/** Eliminar palabra */
public static boolean remove(String palabra) {
	return palabras.remove(palabra.toUpperCase());
}
/** Carga los datos desde el fichero {@link #RUTA} */
public static Boolean load() {
	BufferedReader br = null;
	String palabra;
	try {
		br = new BufferedReader(new FileReader(RUTA));

		while ((palabra = br.readLine()) ≠ null) {
			add(palabra);
		}

		return true;
	} catch (IOException e) {
		e.printStackTrace();
	} finally {
		try {
			if (br ≠ null) {
				br.close();
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	return false;
}
/** Guarda las palabras en el fichero {@link #RUTA} */
public static void save() {
	BufferedWriter bw = null;
	try {
		bw = new BufferedWriter(new FileWriter(RUTA));

		for (String palabra : palabras) {
			bw.write(palabra + "\n");
		}
	} catch (IOException e) {
		e.printStackTrace();
	} finally {
		try {
			if (bw ≠ null) {
				bw.close();
				System.out.printf("Guardando las palabras al fichero %s...\n", RUTA);
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
@Override
public int hashCode() {
	int result = dorsal;
	result = 31 * result;
	return result;
}
@Override
public boolean equals(Object o) {
	if (this == o) return true;
	if (o == null || getClass() ≠ o.getClass()) return false;

	Corredor corredor = (Corredor) o;

	return dorsal == corredor.dorsal;
}
/** Guarda los datos de los {@link Cliente} de la {@link Clinica} en un archivo binario. */
private static void salir() {
	ObjectOutputStream objectOutputStream = null;

	if (!new File("exportaciones").exists()) {
		if (new File("exportaciones").mkdir()) System.out.println("Creando la carpeta exportaciones...");
	}

	try {
		objectOutputStream = new ObjectOutputStream(new FileOutputStream("exportaciones/clientes.dat"));

		for (Cliente cliente : Clinica.getClientes()) {
			objectOutputStream.writeObject(cliente);
		}
	} catch (IOException ex) {
		ex.printStackTrace();
	} finally {
		try {
			if (objectOutputStream ≠ null) objectOutputStream.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
/**
 * Importa los datos de los {@link Cliente} desde un archivo binario.
 *
 * @param exp Archivo binario con los datos.
 */
private static void importarInfoClientes(File exp) {
	ObjectInputStream objectInputStream = null;
	Cliente thisCliente;
	ArrayList<Cliente> clientes = new ArrayList<>();

	if (!exp.exists()) {
		System.out.println("No se puede leer el archivo " + exp + ", no existe.");
	} else {
		try {
			objectInputStream = new ObjectInputStream(new BufferedInputStream(new FileInputStream(exp)));

			while ((thisCliente = (Cliente) objectInputStream.readObject()) ≠ null) {
				clientes.add(thisCliente);
			}
		} catch (EOFException ignored) {
			System.out.println("Importando clientes desde " + exp + "...");
		} catch (IOException | ClassNotFoundException ex) {
			ex.printStackTrace();
		} finally {
			try {
				if (objectInputStream ≠ null) objectInputStream.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}
}
```

```java
private static void palabraToMayus(String palabra, File f) {
	RandomAccessFile fichero = null;
	boolean finPalabra = false;
	boolean EOF = false;
	boolean palabraIgual;
	long puntero = 0;
	int indice = 0;
	char c;
	char[] letras = new char[palabra.length()];

	try {
		fichero = new RandomAccessFile(f, "rw");
		do {
			try {
				/* Se lee un caracter */
				c = (char) fichero.readByte();
				/* Si el puntero no está definido */
				if (puntero == 0) {
					puntero = fichero.getFilePointer() - 1;
				}
				/* Se añaden las letras de la palabra al array, una por vuelta de bucle */
				if (indice < letras.length) {
					letras[indice] = c;
					indice++;
				}
				/* Si se encuentra un espacio */
				if (c == ' ') {
					finPalabra = true;
				}
				/* Si ha terminado la palabra */
				if (finPalabra) {
					palabraIgual = isPalabraIgual(palabra, letras);
					if (palabraIgual) {
						remplazarPalabra(fichero, puntero, letras);
					}
					/* Reinicio de variables */
					finPalabra = false;
					puntero = 0;
					indice = 0;
				}
			} catch (EOFException e) {
				EOF = true;
				/* Para la última palabra del fichero */
				palabraIgual = isPalabraIgual(palabra, letras);
				if (palabraIgual) {
					remplazarPalabra(fichero, puntero, letras);
				}
			}
		} while (!EOF);
	} catch (IOException e) {
		e.printStackTrace();
	} finally {
		try {
			if (fichero ≠ null) {
				fichero.close();
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
/** Se situa en la posición del puntero dentro de un {@link RandomAccessFile} y escribe un char[]. */
private static void remplazarPalabra(RandomAccessFile fichero, long puntero, char[] letras) throws IOException {
	/* Apuntamos al principio de la palabra */
	fichero.seek(puntero);
	for (char letra : letras) {
		/* Escribimos las letras en mayúsculas */
		fichero.writeByte(Character.toUpperCase(letra));
	}
}
```

ENUMERADOS

`public final String name()`
Devuelve un String con el nombre de la constante que contiene tal y como aparece en la declaración.

`public final int ordinal()`
Devuelve un entero con la posición de la constante según está declarada. A la primera constante le corresponde la posición cero.

`public static enumConstant valueOf(String s)`
Devuelve la constante que coincide exactamente con el String que recibe como parámetro.

`public static enumConstant [] values()`
Devuelve un array que contiene todas las constantes de la enumeración en el orden en que se han declarado. Se suele usar en bucles for each para recorrer el enum.