Código

EJERCICIO 1

```
public class Ejercicio1 {
	/*Dada la siguiente definición recursiva de la función f (que toma como
entrada 3 números enteros positivos y devuelve una cadena):

siendo + un operador que representa la concatenación de cadenas, y toString(i) un
método que devuelve una cadena a partir de un entero. Al llevar a cabo la
implementación, para el tratamiento de cadenas se recomienda hacer uso de
String.format.

Proporcione una solución iterativa usando while, una recursiva no
final, una recursiva final, y una en notación funcional*/

public static String fIt (Integer a, Integer b, Integer c) {
	String ac = "";

	while(a+b+c > 3) {
		if(a<3 && b<3 && c<3) {
			ac =  ac + "(" + Integer.toString(a*b*c) +")";
			break;
		}
	if(a<5 || b<5 || c<5) {

		ac =  ac + "(" + Integer.toString(a+b+c) +")";

			break;
	}
	if (a%2==0 && b%2==0 && c%2==0 ) {
		ac = ac + Integer.toString(a*b*c);
		a= a/2;
		b= b-2;
		c= c/2;
	}else {
		ac =  ac + Integer.toString(a+b+c);
		a= a/3;
		b= b-3;
		c= c/3;
		}
	}
	return ac;
}
// Recursivo no final
public static String fRNF(Integer a, Integer b, Integer c) {
	String ac ="";

	if(a<3 && b<3 && c<3) {
		return  ac + "(" + Integer.toString(a*b*c) +")";
	}
	if(a<5 || b<5 || c<5) {
		return   ac +"(" + Integer.toString(a+b+c) +")";
	}
	if (a%2==0 && b%2==0 && c%2==0) {
		ac =   Integer.toString(a*b*c) + fRNF(a/2,b-2,c/2);
	}else {
		ac =  Integer.toString(a+b+c) + fRNF(a/3,b-3,c/3) ;
```

```
	}
	return ac;
}

//Recursivo final

public static String fR(Integer a, Integer b, Integer c) {
	return RAux(a,b,c, "");
}

public static String RAux(Integer a, Integer b, Integer c, String ac) {
	// TODO Auto-generated method stub
	if(a<3 && b<3 && c<3) {
		 ac = ac + "(" + Integer.toString(a*b*c) +")";
		return   ac;

	}
if(a<5 || b<5 || c<5) {
	return  ac + "(" + Integer.toString(a+b+c) +")";

}
if (a%2==0 && b%2==0 && c%2==0 ) {
	return RAux(a/2,b-2,c/2, ac + Integer.toString(a*b*c));
}else {
	return RAux(a/3,b-3,c/3, ac + Integer.toString(a+b+c));
	}
}

//funcional

public static String fF(Integer a, Integer b, Integer c) {
	ej1 r =  ej1.of(a, b, c, "");
	 Optional<ej1> st= Stream.iterate( r, t-> t.next())
			.filter(x->r.esCB(x)).findFirst();
			return	st.isPresent()? casoBase(st.get()): "-1";
}

public static String casoBase( ej1 r) {
	String res="";
	if(r.a()<3 && r.b()<3 && r.c()<3){
		res = r.s() + "(" + Integer.toString(r.a()*r.b()*r.c()) +")";

	}else if (r.a()<5 || r.b()<5 || r.c()<5) {
		res = r.s() + "(" + Integer.toString(r.a()+r.b()+r.c()) +")";
	}
	return res;
}
public static record  ej1(Integer a, Integer b, Integer c, String s) {
	public static ej1 of(Integer a, Integer b, Integer c, String s) {
		return new ej1(a, b, c, s);
	}
	public ej1 next() {
		ej1 res;
		if (a%2==0 && b%2==0 && c%2==0 ) {
			res= ej1.of( a/2, b-2, c/2, s() + Integer.toString(a*b*c)) ;
		}else {
			res= ej1.of(a/3, b-3, c/3,s() +Integer.toString(a+b+c));
		}
		return res;
	}
	public Boolean esCB(ej1 r) {
		Boolean b = (r.a<3 && r.b<3 && r.c<3) || (r.a<5 || r.b<5 || r.c<5);

		return b;
	}
}
```

EJERCICIO 2

```
public class Ejercicio2 {
	/*Dada una matriz de n x n cadena de caracteres (con n=2m; siendo m un número entero
mayor que 0), devolver una lista de cadenas de caracteres que incluya las cadenas que se
forman uniendo las 4 cadenas de las 4 esquinas de la matriz principal, y de cada una de
sus 4 submatrices, y así sucesivamente hasta llegar a una matriz de 2x2. El orden en el
que se unen las cadenas de las esquinas es: superior izquierda, superior derecha, inferior
izquierda, e inferior derecha.

proporcione una solución recursiva.*/


	public static List<String> ej2(Matrix<String> m) {

		return ej2Aux(m, new ArrayList<String>());
	}

	public static List<String> ej2Aux(Matrix<String> m, List<String> ls) {
		// TODO Auto-generated method stub

		if(m.area() >4) {
			String s = "";
			for(String n :  m.corners()) {
				s= s+n;
			}
			ls.add(s);
			View4<Matrix<String>> v = m.views4();
			ej2Aux(v.a(),ls);
			ej2Aux(v.b(),ls);
			ej2Aux(v.c(),ls);
			ej2Aux(v.d(),ls);
		}else {
			String s2 = "";
			for(String n2: m.corners()) {
				s2=s2+n2;
			}
			ls.add(s2);
		}
		return ls;
	}


}
```

EJERCICIO 3

```
public static IntegerSet ej3(List<Integer> ls, Integer a, Integer b) {

		return ej3Aux(ls,IntegerSet.empty(),0,ls.size()-1,a,b);
	}
	public static IntegerSet ej3Aux(List<Integer> ls,IntegerSet res, Integer inC,
Integer inF, Integer start, Integer end) {
		if((end-start)==1) {
			res = casobase(ls,res,inC,inF,start,end);
		}else {
			Integer pivot=(inC+inF)/2;
			IntegerSet a=ej3Aux(ls,res,inC,pivot,start,end);
		    IntegerSet b=ej3Aux(ls,res,pivot,inF,start,end);

		    res = res.union(a).union(b);

		}
		return res;
	}

	public static IntegerSet casobase(List<Integer> ls,IntegerSet res, Integer
inC, Integer inF, Integer start, Integer end) {
		// TODO Auto-generated method stub
		if((inC-inF==1)) {
			if(end>ls.get(inC) && ls.get(inC)>=start) {
				res.add(ls.get(inC));
			}
			if(end>ls.get(inF) && ls.get(inF)>=start) {
				res.add(ls.get(inF));
			}
		}

		return res;
}
```

EJERCICIO 4

```
public class Ejercicio2 {
	/*Dada una matriz de n x n cadena de caracteres (con n=2m; siendo m un número
entero mayor que 0), devolver una lista de cadenas de caracteres que incluya las
cadenas que se forman uniendo las 4 cadenas de las 4 esquinas de la matriz principal,
y de cada una de sus 4 submatrices, y así sucesivamente hasta llegar a una matriz de
2x2. El orden en el que se unen las cadenas de las esquinas es: superior izquierda,
superior derecha, inferior
izquierda, e inferior derecha.proporcione una solución recursiva.*/

	public static List<String> ej2(Matrix<String> m) {

		return ej2Aux(m, new ArrayList<String>());
	}

	public static List<String> ej2Aux(Matrix<String> m, List<String> ls) {
		// TODO Auto-generated method stub

		if(m.area() >4) {
			String s = "";
			for(String n :  m.corners()) {
				s= s+n;
			}
			ls.add(s);
			View4<Matrix<String>> v = m.views4();
			ej2Aux(v.a(),ls);
			ej2Aux(v.b(),ls);
			ej2Aux(v.c(),ls);
			ej2Aux(v.d(),ls);
		}else {
			String s2 = "";
			for(String n2: m.corners()) {
				s2=s2+n2;
			}
			ls.add(s2);
		}
		return ls;
	}


}
```

EJERCICIO 5

```
public class Ejercicio5 {
	/*
	 *Diseñar un algoritmo recursivo, con y sin memoria, y posteriormente
encontrar un algoritmo iterativo para la siguiente definición:
siendo a, b y c números enteros positivos
 a+b^2+ 2*c   para    a<3 o b<3 o c<3
 g(a-1, b/2, c/2) + g(a-3, b/3,c/3) para a es multiplo b
 g(a/3,b-3,c-3) + g(a/2,b-2,c-2) en otro caso
proporcione una solución recursiva sin memoria, otra recursiva
con memoria (map), y otra iterativa.
*/
	//recursivo sin memoria
public static Integer ej5RSM(Integer a, Integer b, Integer c) {
		Integer res= 0 ;
		if(a<3 || b<3 || c<3) {
			res= a+b*b+ 2*c;
		}else if(a%b==0) {
			res= ej5RSM(a-1, b/2, c/2) + ej5RSM(a-3, b/3,c/3);
		}else {
			res= ej5RSM(a/3, b-3, c-3) + ej5RSM(a/2, b-2,c-2);
		}

		return res;
	}
	//recursivo con memoria
	public static Integer ej5RCM(Integer a, Integer b, Integer c) {

		return ej5RCMAux(a,b,c, new HashMap<>());
	}



public static Integer ej5RCMAux(Integer a,Integer b,Integer c,Map<IntTrio,Integer> m)
{
		Integer res= 0 ;
		if(m.containsKey(IntTrio.of(a, b, c))){
			return m.get(IntTrio.of(a, b, c));
		}else {
		if(a<3 || b<3 || c<3) {
			res= a+b*b+ 2*c;
			m.put(IntTrio.of(a, b, c),res);
			return res;
		}
		if(a%b==0) {
			res= ej5RCMAux(a-1, b/2, c/2,m) + ej5RCMAux(a-3, b/3,c/3,m);
		}else {
			res= ej5RCMAux(a/3, b-3, c-3,m) + ej5RCMAux(a/2, b-2,c-2,m);
			}
		}
		return res;
	}

//iterativo
	public static Integer ej5It(Integer a, Integer b, Integer c) {
	Integer res= 0 ;
	Map<IntTrio,Integer> m = new HashMap<>();
	int i = 0;


	if(m.containsKey(IntTrio.of(a, b, c))){
		return m.get(IntTrio.of(a, b, c));
	}
	while(i<=a) {
		int j = 0;
```

```
                while(j<=b) {
                        int k=0;
                        while(k<=c){

                        if(i<3 || j<3 || k<3) {
                                        res= i+(j*j)+ 2*k;
                        }else if(i%j==0) {
                                res=m.get(IntTrio.of(i-1, j/2, k/2)) +
                                    m.get(IntTrio.of(i-3, j/3,k/3));;
                                        //a-1, b/2, c/2 + a-3, b/3,c/3;

                                }else {
                                        res= m.get(IntTrio.of(i/3, j-3, k-3)) +
                                                m.get(IntTrio.of(i/2, j-2,k-2));
                                               //(a/3, b-3, c-3) + (a/2, b-2,c-2);
                                               }
                                               m.put(IntTrio.of(i, j, k), res);
                                               k++;
                                        }
                                        j++;
                                }
                                i++;
                        }
                return res;
        }
}
```

CLASE TEST

```
public static void main(String[] args) {
		// TODO Auto-generated method stub
		test1();
		test2();
		test3();
		test4();
		test5();

	}
```

EJERCICIO 1

```
public static void test1() {
		List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej1DatosEntrada.txt");
System.out.println("#####################################################");
System.out.println("#                   Ejercicio 1                     #");
System.out.println("#         ficheros/PI2Ej1DatosEntrada.txt           #");
System.out.println("#####################################################");
   for (String linea : filas) {
	System.out.println("Entrada"+"["+ linea +"]");
	String[] partes = linea.split(",");
		Integer a = Integer.parseInt(partes[0]);
		Integer b = Integer.parseInt(partes[1]);
		Integer c = Integer.parseInt(partes[2]);
	System.out.println(String.format("Recursivo no final: %s",
	Ejercicio1.fRNF(a, b, c)));
	System.out.println(String.format("Iterativo:          %s",
	Ejercicio1.fIt(a, b, c)));
	System.out.println(String.format("Recursivo final:    %s",
	Ejercicio1.fR(a, b, c)));
	System.out.println(String.format("Funcional:          %s",
	Ejercicio1.fF(a, b, c)));
		}
	System.out.println("#####################################################");

	}
```

EJERCICIO 2

```
public static void test2() {
		//List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej2DatosEntrada1.txt");// FICHERO 1
		List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej2DatosEntrada2.txt");// FICHERO 2

	System.out.println("#####################################################");
	System.out.println("#                   Ejercicio 2                     #");
	//System.out.println("#         ficheros/PI2Ej2DatosEntrada1.txt          #");
	System.out.println("#         ficheros/PI2Ej2DatosEntrada1.txt        #");
	System.out.println("#####################################################");
	//creo una lista de listas que mastarde se convertirá en una matriz
	List<List<String>> f2 = new ArrayList<>();
	for(String linea : filas ) {
		// lista que se crea a partir del array de String
		List<String> ls = new ArrayList<>();
		String[] partes = linea.split(" ");
		for(String parte : partes) {
			ls.add(parte);
		}
		//tras  terminar la lista se añade a la lista de listas
		f2.add(ls);
	}
```

```
        System.out.println("Entrada : "+ f2);
        Integer tam= f2.size();
        String[][] mdata = new String [tam][tam];
        for(int i=0;i<tam;i++) {
                for(int j=0;j<tam;j++) {
                        String a= f2.get(i).get(j);

                        mdata[i][j]=a;
                        }
                }
        Matrix<String> m =Matrix.of(mdata);
        System.out.println(String.format("Recursivo :    %s", Ejercicio2.ej2(m)));

        System.out.println("##################################################");
        }
```

EJERCICIO 3

```
public static void test3() {
                List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej3DatosEntrada.txt");

        System.out.println("##################################################");
        System.out.println("#                   Ejercicio 3                  #");
        System.out.println("#         ficheros/PI2Ej3DatosEntrada.txt       #");
        System.out.println("##################################################");
        for (String linea: filas) {
                String [] partes = linea.split("#");
                String [] partes2 = partes[1].split(",");
                String [] partes3 = partes[0].split(",");
                Integer a = Integer.parseInt(partes2[0]);
                Integer b = Integer.parseInt(partes2[1]);
                List<Integer> ls = new ArrayList<>();
                for(String parte : partes3) {
                        Integer z = Integer.parseInt(parte);
                        ls.add(z);
                }
                System.out.println("Entrada: " + ls);
                System.out.println("Rango: " + "[" + a + "," + b+ ")");
                System.out.println("Recursivo Multiple: " + Ejercicio3.ej3(ls, a, b) );


        }
```

EJERCICIO 4

```
public static void test4() {
                List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej4DatosEntrada.txt");

        System.out.println("##################################################");
        System.out.println("#                   Ejercicio 4                  #");
        System.out.println("#         ficheros/PI2Ej4DatosEntrada.txt        #");
        System.out.println("##################################################");
        for (String linea : filas) {
```

```
                System.out.println("Entrada (a,b,c): "+"("+ linea +")");
                String[] partes = linea.split("n=");
                        Integer a = Integer.parseInt(partes[1]);
                        System.out.println(String.format("Recursivo Sin Mem: %s",
Ejercicio4.ej4RSM(a)));
                        System.out.println(String.format("Recursivo Con Mem: %s",
Ejercicio4.ej4RCM(a)));
                        System.out.println(String.format("Iterativo:         %s",
Ejercicio4.ej4It(a)));
                        }
        System.out.println("#################################################");
        }
```

EJERCICIO 5

```
public static void test5() {
                List<String> filas =
Files2.linesFromFile("./ficheros/PI2Ej5DatosEntrada.txt");

        System.out.println("#################################################");
        System.out.println("#                   Ejercicio 5                 #");
        System.out.println("#         ficheros/PI2Ej5DatosEntrada.txt       #");
        System.out.println("#################################################");
        for (String linea : filas) {
                System.out.println("Entrada (a,b,c): "+"("+ linea +")");
                String[] partes = linea.split(",");
                        Integer a = Integer.parseInt(partes[0]);
                        Integer b = Integer.parseInt(partes[1]);
                        Integer c = Integer.parseInt(partes[2]);
                        System.out.println(String.format("Recursivo Sin Mem: %s",
Ejercicio5.ej5RSM(a, b, c)));
                        System.out.println(String.format("Recursivo Con Mem: %s",
Ejercicio5.ej5RCM(a, b, c)));
                        System.out.println(String.format("Iterativo:         %s",
Ejercicio5.ej5It(a, b, c)));
                        }
        System.out.println("#################################################");
        }
```

SALIDAS

EJERCICIO 1

```
#####################################################
#                   Ejercicio 1                     #
#         ficheros/PI2Ej1DatosEntrada.txt           #
#####################################################
Entrada[20,40,80]
Recursivo no final: 640001520061(40)
Iterativo:          640001520061(40)
Recursivo final:    640001520061(40)
Funcional:          640001520061(40)
Entrada[40,20,10]
Recursivo no final: 800043(22)
Iterativo:          800043(22)
Recursivo final:    800043(22)
Funcional:          800043(22)
Entrada[5,25,125]
Recursivo no final: 155(64)
Iterativo:          155(64)
Recursivo final:    155(64)
Funcional:          155(64)
Entrada[125,5,25]
Recursivo no final: 155(51)
Iterativo:          155(51)
Recursivo final:    155(51)
Funcional:          155(51)
Entrada[100,50,1]
Recursivo no final: (151)
Iterativo:          (151)
Recursivo final:    (151)
Funcional:          (151)
Entrada[1,50,200]
Recursivo no final: (251)
Iterativo:          (251)
Recursivo final:    (251)
Funcional:          (251)
#####################################################
```

EJERCICIO 2

A - FICHEROS/PI2EJ2DATOSENTRADA1.TXT

```
##################################################
#                   Ejercicio 2                  #
#         ficheros/PI2Ej2DatosEntrada2.txt       #
##################################################
Entrada :
[[abstract_, assert_, boolean_, break_, byte_, case_, catch_, char_],
[class_, continue, default_, do, double_, else, enum_, exports],
[extends_, final_, finally_, float_, for_, if_, implements_, import_],
[instanceof_, int, interface_, long, module_, native, new_, non-sealed],
[package_, permits_, private_, protected_, public_, sealed_, record_,
return_],
[short_, static, strictfp_, super, switch_, synchronized, this_, throw],
[throws_, transient_, try_, void_, volatile_, while_, var_, yield_],
[parte_, comun, datos_, compartidos, grafos_, solve, geneticos_, ejemplos]]

Recursivo :
[abstract_char_parte_ejemplos,
abstract_break_instanceof_long,
abstract_assert_class_continue,
 boolean_break_default_do,
 extends_final_instanceof_int,
finally_float_interface_long,
byte_char_module_non-sealed,
 byte_case_double_else,
catch_char_enum_exports,
 for_if_module_native,
 implements_import_new_non-sealed,
 package_protected_parte_compartidos,
 package_permits_short_static,
 private_protected_strictfp_super,
 throws_transient_parte_comun,
 try_void_datos_compartidos,
 public_return_grafos_ejemplos,
 public_sealed_switch_synchronized,
 record_return_this_throw,
 volatile_while_grafos_solve,
var_yield_geneticos_ejemplos]
##################################################
```

B - FICHEROS/PI2EJ2DATOSENTRADA2.TXT

```
##################################################
#                  Ejercicio 2                   #
#         ficheros/PI2Ej2DatosEntrada2.txt       #
##################################################
Entrada : [[Para, abordar, el, diseño, de, algoritmos, es, necesario],
[tener, asimilados, los, elementos, de, la, programación, en],
[algún, lenguaje., Para, seguir, el, contenido, hace, falta],
[conocer, el, lenguaje, Java, y, sus, peculiaridades., Aprenderemos],
[las, técnicas, para, el, diseño, de, algoritmos, iterativos,],
[las, técnicas, de, diseño, de, algoritmos, recursivos,, su],
[análisis, y, las, transformaciones, de, un, tipo, de],
[algoritmo, en, otros., Al, final, se, incluyen, ejemplos]]
Recursivo :
[Paranecesarioalgoritmoejemplos,
ParadiseñoconocerJava,
Paraabordartenerasimilados,
eldiseñoloselementos,
algúnlenguaje.conocerel,
ParaseguirlenguajeJava,
denecesarioyAprenderemos,
dealgoritmosdela,
esnecesarioprogramaciónen,
elcontenidoysus,
hacefaltapeculiaridades.Aprenderemos,
laselalgoritmoAl,
lastécnicaslastécnicas,
paraeldediseño,
análisisyalgoritmoen,
lastransformacionesotros.Al,
diseñoiterativos,
finalejemplos,
diseñodedealgoritmos,
algoritmositerativos,
recursivos,su,
deunfinalse,
tipodeincluyenejemplos]
##################################################
```

EJERCICIO 3

```
#####################################################
#                   Ejercicio 3                     #
#         ficheros/PI2Ej3DatosEntrada.txt           #
#####################################################
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [2,7)
Recursivo Multiple: {2,3,4,5,6}
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [0,1)
Recursivo Multiple: {0}
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [9,10)
Recursivo Multiple: {9}
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [11,20)
Recursivo Multiple: {}
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [8,20)
Recursivo Multiple: {8,9,10}
Entrada: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Rango: [0,5)
Recursivo Multiple: {0,1,2,3,4}
#####################################################
```

EJERCICIO 4

```
#####################################################
#                   Ejercicio 4                     #
#         ficheros/PI2Ej4DatosEntrada.txt           #
#####################################################
Entrada (a,b,c): (n=5)
Recursivo Sin Mem: 452
Recursivo Con Mem: 452
Iterativo:         452
Entrada (a,b,c): (n=10)
Recursivo Sin Mem: 271200
Recursivo Con Mem: 271200
Iterativo:         271200
Entrada (a,b,c): (n=15)
Recursivo Sin Mem: 160269440
Recursivo Con Mem: 160269440
Iterativo:         160269440
Entrada (a,b,c): (n=20)
Recursivo Sin Mem: 94705116032
Recursivo Con Mem: 94705116032
Iterativo:         94705116032
Entrada (a,b,c): (n=25)
Recursivo Sin Mem: 55962400789504
Recursivo Con Mem: 55962400789504
Iterativo:         55962400789504
Entrada (a,b,c): (n=30)
Recursivo Sin Mem: 33068860966434816
Recursivo Con Mem: 33068860966434816
Iterativo:         33068860966434816
#####################################################
```

EJERCICIO 5

```
#####################################################
#                   Ejercicio 5                     #
#         ficheros/PI2Ej5DatosEntrada.txt           #
#####################################################
Entrada (a,b,c): (20,10,5)
Recursivo Sin Mem: 76
Recursivo Con Mem: 76
Iterativo:         76
Entrada (a,b,c): (40,20,10)
Recursivo Sin Mem: 201
Recursivo Con Mem: 201
Iterativo:         201
Entrada (a,b,c): (80,40,20)
Recursivo Sin Mem: 1860
Recursivo Con Mem: 1860
Iterativo:         1860
Entrada (a,b,c): (20,40,80)
Recursivo Sin Mem: 8658
Recursivo Con Mem: 8658
Iterativo:         8658
Entrada (a,b,c): (10,20,40)
Recursivo Sin Mem: 1187
Recursivo Con Mem: 1187
Iterativo:         1187
Entrada (a,b,c): (5,10,20)
Recursivo Sin Mem: 186
Recursivo Con Mem: 186
Iterativo:         186
#####################################################
```