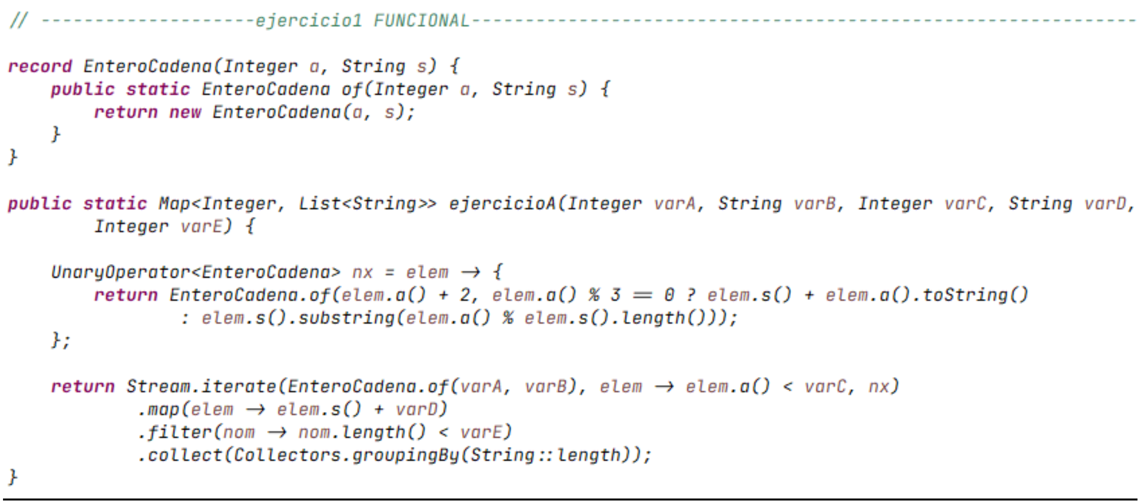
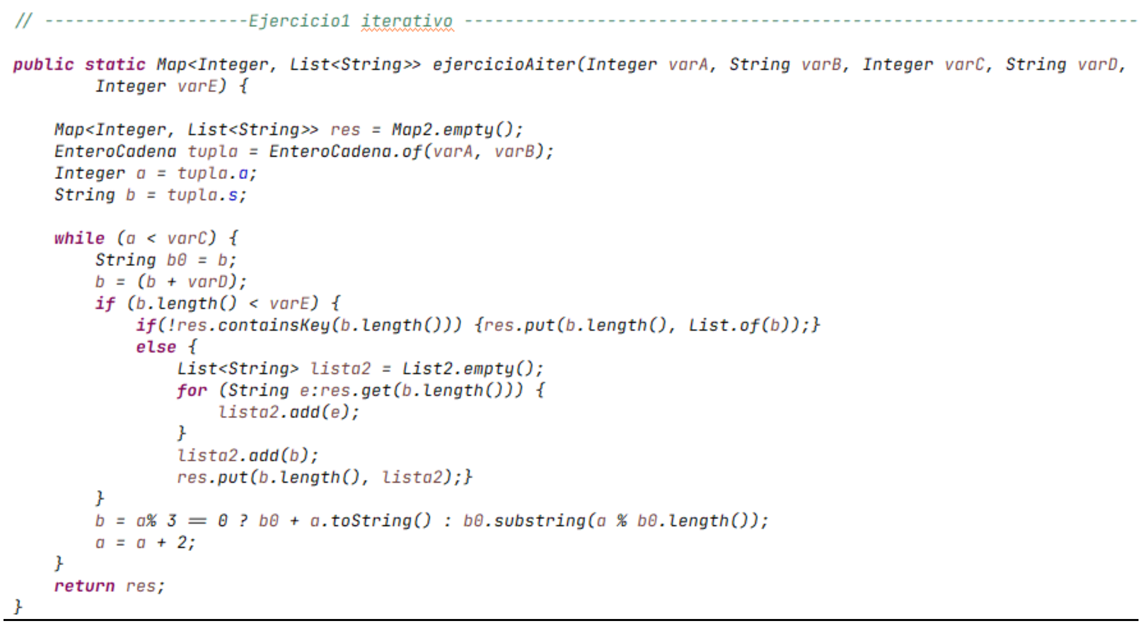
MARCOS A RODRÍGUEZ GARCÍA

MEMORIA PI1 MARRODGAR60

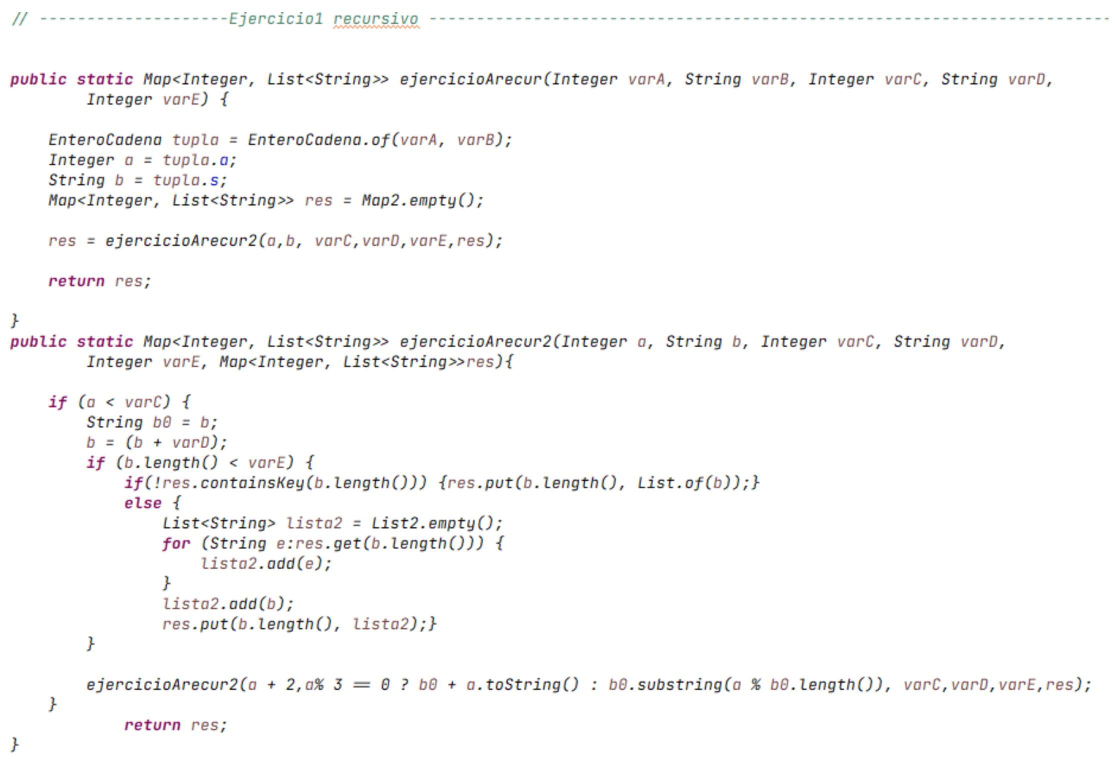
EJERCICIO 1



```java
// --------------------ejercicio1 FUNCIONAL------------------------------------------------------------------

record EnteroCadena(Integer a, String s) {
    public static EnteroCadena of(Integer a, String s) {
        return new EnteroCadena(a, s);
    }
}

public static Map<Integer, List<String>> ejercicioA(Integer varA, String varB, Integer varC, String varD,
        Integer varE) {

    UnaryOperator<EnteroCadena> nx = elem -> {
        return EnteroCadena.of(elem.a() + 2, elem.a() % 3 == 0 ? elem.s() + elem.a().toString()
                : elem.s().substring(elem.a() % elem.s().length()));
    };

    return Stream.iterate(EnteroCadena.of(varA, varB), elem -> elem.a() < varC, nx)
            .map(elem -> elem.s() + varD)
            .filter(nom -> nom.length() < varE)
            .collect(Collectors.groupingBy(String::length));
}
```

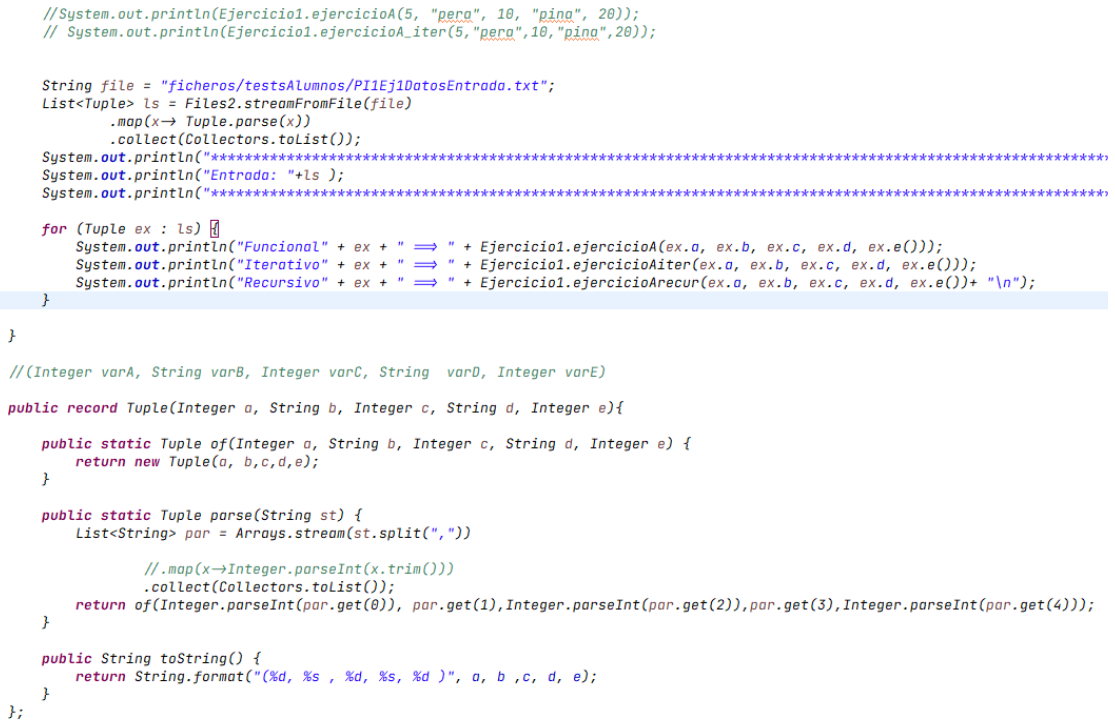


```java
// -------------------Ejercicio1 iterativo ------------------------------------------------------------------

public static Map<Integer, List<String>> ejercicioAiter(Integer varA, String varB, Integer varC, String varD,
        Integer varE) {

    Map<Integer, List<String>> res = Map2.empty();
    EnteroCadena tupla = EnteroCadena.of(varA, varB);
    Integer a = tupla.a;
    String b = tupla.s;

    while (a < varC) {
        String b0 = b;
        b = (b + varD);
        if (b.length() < varE) {
            if(!res.containsKey(b.length())) {res.put(b.length(), List.of(b));}
            else {
                List<String> lista2 = List2.empty();
                for (String e:res.get(b.length())) {
                    lista2.add(e);
                }
                lista2.add(b);
                res.put(b.length(), lista2);}
        }
        b = a% 3 == 0 ? b0 + a.toString() : b0.substring(a % b0.length());
        a = a + 2;
    }
    return res;
}
```



```java
// -------------------Ejercicio1 recursivo ---------------------------------------------------------------------------

public static Map<Integer, List<String>> ejercicioArecur(Integer varA, String varB, Integer varC, String varD,
        Integer varE) {

    EnteroCadena tupla = EnteroCadena.of(varA, varB);
    Integer a = tupla.a;
    String b = tupla.s;
    Map<Integer, List<String>> res = Map2.empty();

    res = ejercicioArecur2(a,b, varC,varD,varE,res);

    return res;

}
public static Map<Integer, List<String>> ejercicioArecur2(Integer a, String b, Integer varC, String varD,
        Integer varE, Map<Integer, List<String>>res){

    if (a < varC) {
        String b0 = b;
        b = (b + varD);
        if (b.length() < varE) {
            if(!res.containsKey(b.length())) {res.put(b.length(), List.of(b));}
            else {
                List<String> lista2 = List2.empty();
                for (String e:res.get(b.length())) {
                    lista2.add(e);
                }
                lista2.add(b);
                res.put(b.length(), lista2);}
        }

        ejercicioArecur2(a + 2,a% 3 == 0 ? b0 + a.toString() : b0.substring(a % b0.length()), varC,varD,varE,res);
    }
        return res;
}
```

EJERCICIO 1 TEST



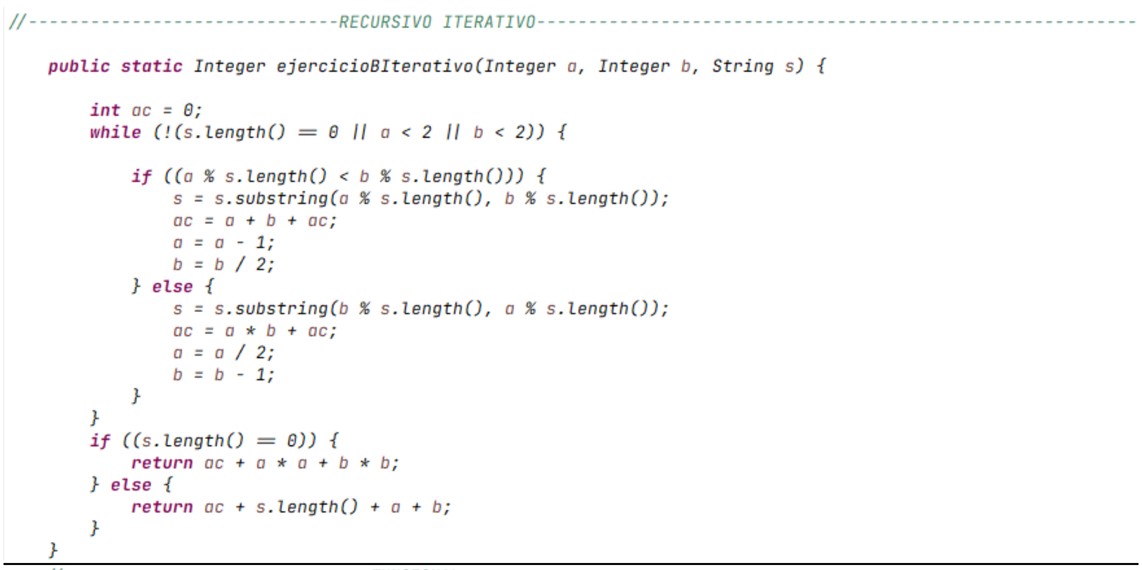
```java
    //System.out.println(Ejercicio1.ejercicioA(5, "pera", 10, "pina", 20));
    // System.out.println(Ejercicio1.ejercicioA_iter(5,"pera",10,"pina",20));


    String file = "ficheros/testsAlumnos/PI1Ej1DatosEntrada.txt";
    List<Tuple> ls = Files2.streamFromFile(file)
            .map(x-> Tuple.parse(x))
            .collect(Collectors.toList());
    System.out.println("****************************************************************************************************
    System.out.println("Entrada: "+ls );
    System.out.println("****************************************************************************************************

    for (Tuple ex : ls) {
        System.out.println("Funcional" + ex + " => " + Ejercicio1.ejercicioA(ex.a, ex.b, ex.c, ex.d, ex.e()));
        System.out.println("Iterativo" + ex + " => " + Ejercicio1.ejercicioAiter(ex.a, ex.b, ex.c, ex.d, ex.e()));
        System.out.println("Recursivo" + ex + " => " + Ejercicio1.ejercicioArecur(ex.a, ex.b, ex.c, ex.d, ex.e())+ "\n");
    }

}

//(Integer varA, String varB, Integer varC, String  varD, Integer varE)

public record Tuple(Integer a, String b, Integer c, String d, Integer e){

    public static Tuple of(Integer a, String b, Integer c, String d, Integer e) {
        return new Tuple(a, b,c,d,e);
    }

    public static Tuple parse(String st) {
        List<String> par = Arrays.stream(st.split(","))

                //.map(x->Integer.parseInt(x.trim()))
                .collect(Collectors.toList());
        return of(Integer.parseInt(par.get(0)), par.get(1),Integer.parseInt(par.get(2)),par.get(3),Integer.parseInt(par.get(4)));
    }

    public String toString() {
        return String.format("(%d, %s , %d, %s, %d )", a, b ,c, d, e);
    }
};
```

```
************************************************************************************************************************************************************************************************
Entrada: [(5, java , 10, eclipse, 20 ), (10, interface , 20, class, 30 ), (4, void , 8, return, 16 ), (5, for , 15, while, 25 ), (20, if , 30, else, 40 ), (15, import , 25, static, 50 )]
************************************************************************************************************************************************************************************************

Funcional(5, java , 10, eclipse, 20 ) ==> {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
Iterativo(5, java , 10, eclipse, 20 ) ==> {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}
Recursivo(5, java , 10, eclipse, 20 ) ==> {9=[vaeclipse], 10=[avaeclipse], 11=[javaeclipse]}

Funcional(10, interface , 20, class, 30 ) ==> {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
Iterativo(10, interface , 20, class, 30 ) ==> {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}
Recursivo(10, interface , 20, class, 30 ) ==> {7=[12class], 11=[face12class], 13=[nterfaceclass], 14=[interfaceclass], 15=[nterface12class]}

Funcional(4, void , 8, return, 16 ) ==> {10=[voidreturn, voidreturn]}
Iterativo(4, void , 8, return, 16 ) ==> {10=[voidreturn, voidreturn]}
Recursivo(4, void , 8, return, 16 ) ==> {10=[voidreturn, voidreturn]}

Funcional(5, for , 15, while, 25 ) ==> {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
Iterativo(5, for , 15, while, 25 ) ==> {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}
Recursivo(5, for , 15, while, 25 ) ==> {6=[rwhile, rwhile, 9while], 7=[r9while], 8=[forwhile]}

Funcional(20, if , 30, else, 40 ) ==> {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
Iterativo(20, if , 30, else, 40 ) ==> {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}
Recursivo(20, if , 30, else, 40 ) ==> {6=[ifelse, ifelse, ifelse, 24else], 8=[if24else]}

Funcional(15, import , 25, static, 50 ) ==> {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
Iterativo(15, import , 25, static, 50 ) ==> {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
Recursivo(15, import , 25, static, 50 ) ==> {8=[15static], 10=[1521static], 12=[importstatic], 13=[mport15static], 14=[import15static]}
```

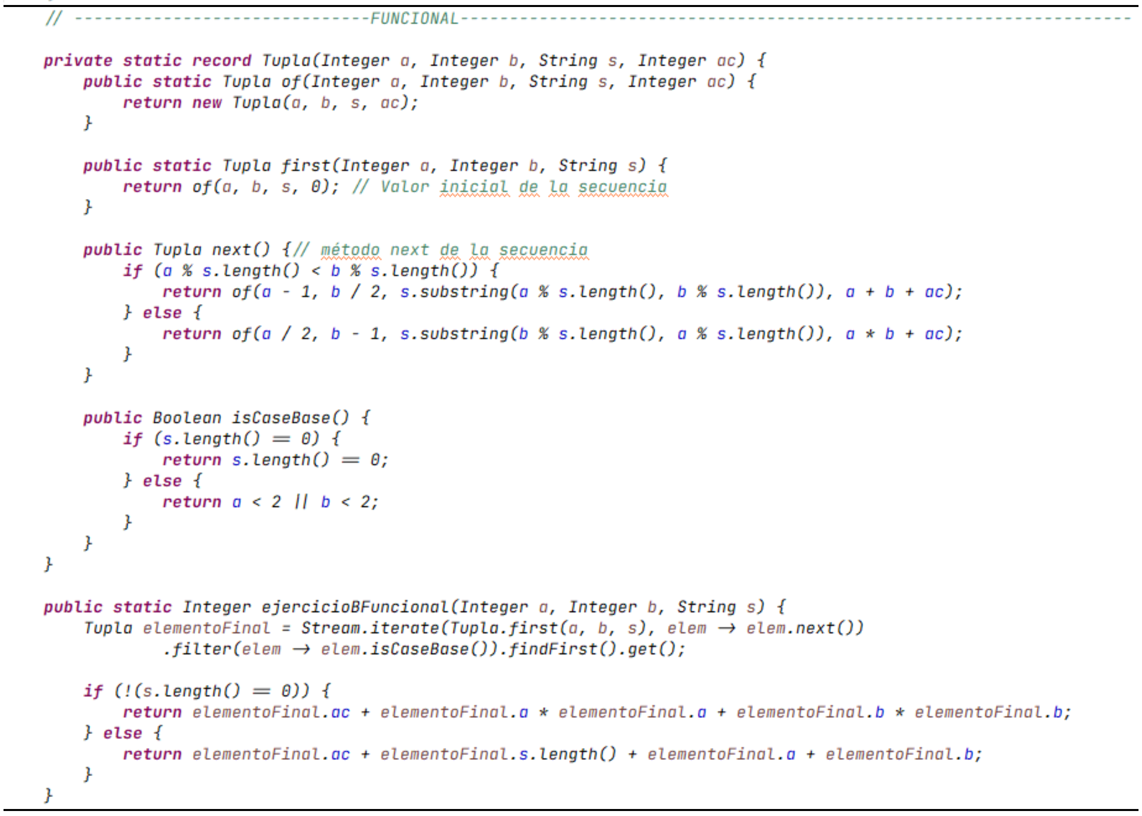
EJERCICIO 2

```java
//------------------------------RECURSIVO NO FINAL---------------------------------------------------------------

    public static Integer ejercicioBRecursivoNoFinal(Integer a, Integer b, String s) {
        int ac = 0;
        if (s.length() == 0) {
            ac = a * a + b * b;
        } else if (a < 2 || b < 2) {
            ac = s.length() + a + b;
        } else if (a % s.length() < b % s.length()) {
            ac = a + b + ejercicioBRecursivoNoFinal(a - 1, b / 2, s.substring(a % s.length(), b % s.length()));
        } else {
            ac = a * b + ejercicioBRecursivoNoFinal(a / 2, b - 1, s.substring(b % s.length(), a % s.length()));
        }
        return ac;
    }
```

```java
//-----------------------------RECURSIVO FINAL-----------------------------------------------------------

    public static Integer ejercicioBRecursivoFinal(Integer a, Integer b, String s) {
        int ac = 0;
        ac = ejercicioBRecursivoFinal(a, b, s, ac);
        return ac;
    }

    public static Integer ejercicioBRecursivoFinal(Integer a, Integer b, String s, Integer ac) {

        if (s.length() == 0) {
            ac = ac + a * a + b * b;
        } else if (a < 2 || b < 2) {
            ac = ac + s.length() + a + b;
        } else if (a % s.length() < b % s.length()) {
            ac = ejercicioBRecursivoFinal(a - 1, b / 2, s.substring(a % s.length(), b % s.length()), a + b + ac);
        } else {
            ac = ejercicioBRecursivoFinal(a / 2, b - 1, s.substring(b % s.length(), a % s.length()), a * b + ac);
        }
        return ac;
    }
```



```java
//----------------------------RECURSIVO ITERATIVO----------------------------------------------------------

    public static Integer ejercicioBIterativo(Integer a, Integer b, String s) {

        int ac = 0;
        while (!(s.length() == 0 || a < 2 || b < 2)) {

            if ((a % s.length() < b % s.length())) {
                s = s.substring(a % s.length(), b % s.length());
                ac = a + b + ac;
                a = a - 1;
                b = b / 2;
            } else {
                s = s.substring(b % s.length(), a % s.length());
                ac = a * b + ac;
                a = a / 2;
                b = b - 1;
            }
        }
        if ((s.length() == 0)) {
            return ac + a * a + b * b;
        } else {
            return ac + s.length() + a + b;
        }
    }
```



```java
//  ----------------------------FUNCIONAL---------------------------------------------------------------

    private static record Tupla(Integer a, Integer b, String s, Integer ac) {
        public static Tupla of(Integer a, Integer b, String s, Integer ac) {
            return new Tupla(a, b, s, ac);
        }

        public static Tupla first(Integer a, Integer b, String s) {
            return of(a, b, s, 0); // Valor inicial de la secuencia
        }

        public Tupla next() {// método next de la secuencia
            if (a % s.length() < b % s.length()) {
                return of(a - 1, b / 2, s.substring(a % s.length(), b % s.length()), a + b + ac);
            } else {
                return of(a / 2, b - 1, s.substring(b % s.length(), a % s.length()), a * b + ac);
            }
        }

        public Boolean isCaseBase() {
            if (s.length() == 0) {
                return s.length() == 0;
            } else {
                return a < 2 || b < 2;
            }
        }
    }

    public static Integer ejercicioBFuncional(Integer a, Integer b, String s) {
        Tupla elementoFinal = Stream.iterate(Tupla.first(a, b, s), elem -> elem.next())
                .filter(elem -> elem.isCaseBase()).findFirst().get();

        if (!(s.length() == 0)) {
            return elementoFinal.ac + elementoFinal.a * elementoFinal.a + elementoFinal.b * elementoFinal.b;
        } else {
            return elementoFinal.ac + elementoFinal.s.length() + elementoFinal.a + elementoFinal.b;
        }
    }
```

EJERCICIO 2 TEST

```java
public class TestEjercicio2 {

    public static void main(String[] args) {
        String file = "ficheros/testsAlumnos/PI1Ej2DatosEntrada.txt";
        List<Tuple> ls = Files2.streamFromFile(file).map(x -> Tuple.parse(x)).collect(Collectors.toList());

        System.out.println(
                "****************************************************************************************************
        System.out.println("Entrada: " + ls);
        System.out.println(
                "****************************************************************************************************
                        + "\n");

        for (Tuple ex : ls) {
            System.out.println(
                    "Recursivo NO Final " + ex + " ==> " + Ejercicio2.ejercicioBRecursivoNoFinal(ex.a, ex.b, ex.s));
            System.out.println(
                    "Recursivo  Final " + ex + " ==> " + Ejercicio2.ejercicioBRecursivoFinal(ex.a, ex.b, ex.s));
            System.out.println("Iterativo " + ex + " ==> " + Ejercicio2.ejercicioBIterativo(ex.a, ex.b, ex.s));
            System.out.println("Funcional " + ex + " ==> " + Ejercicio2.ejercicioBFuncional(ex.a, ex.b, ex.s) + "\n");
        }
    }

    public record Tuple(Integer a, Integer b, String s) {

        public static Tuple of(Integer a, Integer b, String s) {
            return new Tuple(a, b, s);
        }

        public static Tuple parse(String st) {
            List<String> par = Arrays.stream(st.split(",")).map(x -> x.trim()).collect(Collectors.toList());
            return of(Integer.parseInt(par.get(0)), Integer.parseInt(par.get(1)), par.get(2));
        }

        public String toString() {
            return String.format("(%d, %d , %s)", a, b, s);
        }
    }
}
```

```
***********************************************************************************************************************************
Entrada: [(10, 20 , adda), (20, 30 , second course), (30, 40 , analysis), (40, 50 , design), (50, 75 , data), (75, 50 , algorithms)]
***********************************************************************************************************************************

Recursivo NO Final (10, 20 , adda) ==> 623
Recursivo  Final (10, 20 , adda) ==> 623
Iterativo (10, 20 , adda) ==> 623
Funcional (10, 20 , adda) ==> 623

Recursivo NO Final (20, 30 , second course) ==> 950
Recursivo  Final (20, 30 , second course) ==> 950
Iterativo (20, 30 , second course) ==> 950
Funcional (20, 30 , second course) ==> 950

Recursivo NO Final (30, 40 , analysis) ==> 3278
Recursivo  Final (30, 40 , analysis) ==> 3278
Iterativo (30, 40 , analysis) ==> 3278
Funcional (30, 40 , analysis) ==> 3278

Recursivo NO Final (40, 50 , design) ==> 3135
Recursivo  Final (40, 50 , design) ==> 3135
Iterativo (40, 50 , design) ==> 3135
Funcional (40, 50 , design) ==> 3135

Recursivo NO Final (50, 75 , data) ==> 3810
Recursivo  Final (50, 75 , data) ==> 3810
Iterativo (50, 75 , data) ==> 3810
Funcional (50, 75 , data) ==> 3810

Recursivo NO Final (75, 50 , algorithms) ==> 5553
Recursivo  Final (75, 50 , algorithms) ==> 5553
Iterativo (75, 50 , algorithms) ==> 5553
Funcional (75, 50 , algorithms) ==> 5553
```

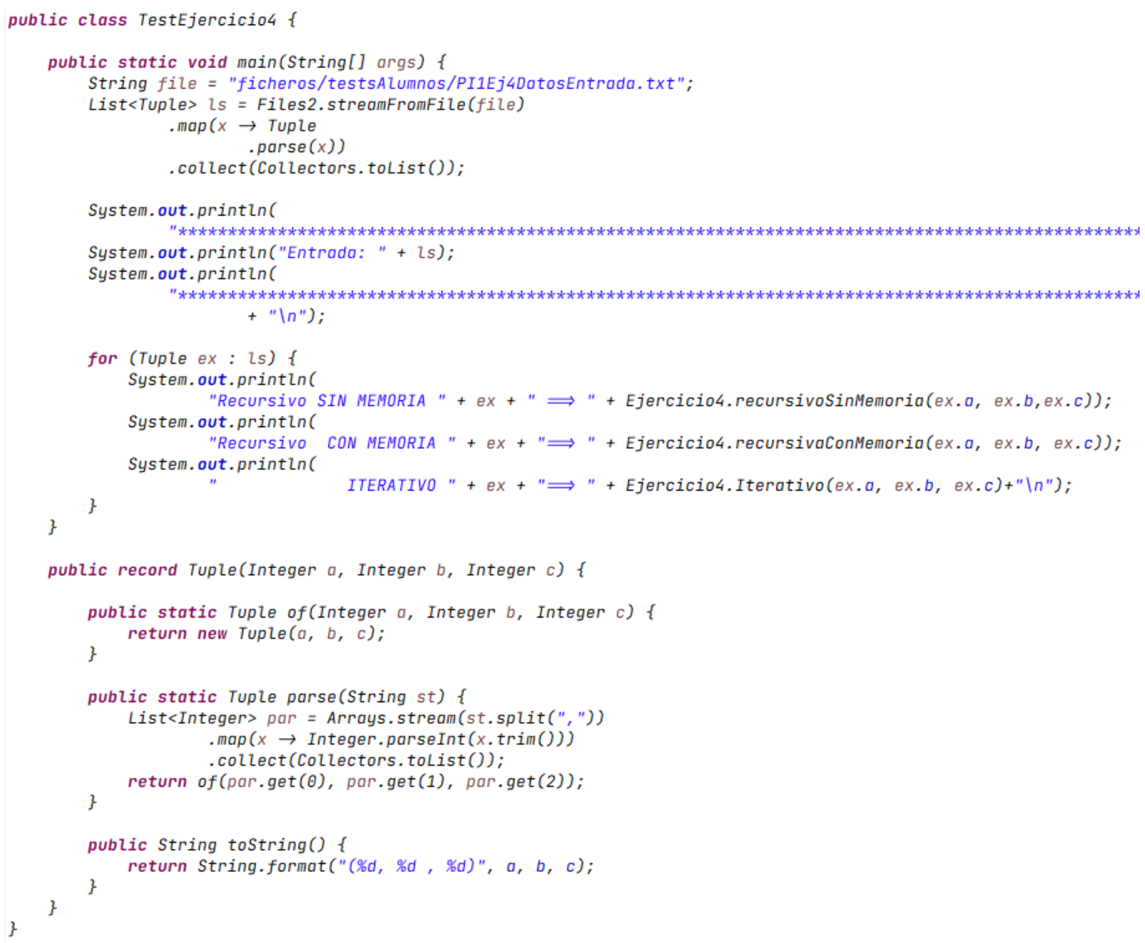
EJERCICIO 3

EJERCICIO 3 TEST

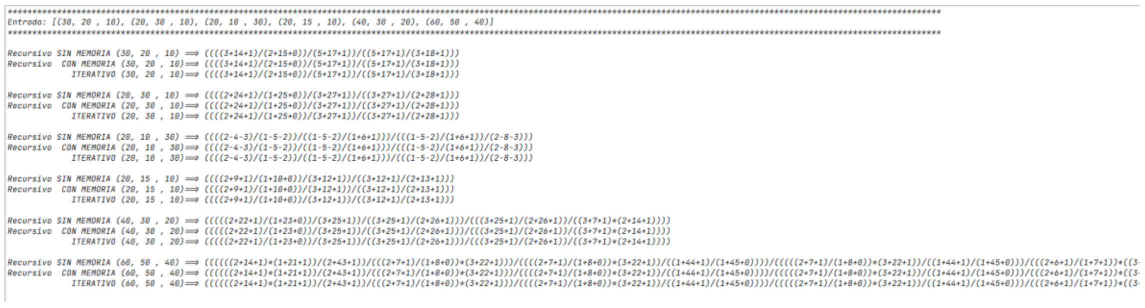
EJERCICIO 4

```java
//---------------------------------RECURSIVA SIN MEMORIA-------------------------------------------------------------------

    public static String recursivoSinMemoria (Integer a, Integer b , Integer c) {
        String ac= "";
        if(a < 2 && b <= 2 || c < 2) {
            ac = String.format("(%d+%d+%d)", a,b,c);
        }else if(a<3 || b<3 && c<3) {
            ac = String.format("(%d-%d-%d)", a,b,c);
            }else if(b%a== 0 && (a%2==0 || b%3 ==0)) {
                ac = String.format("(%s*%s)", recursivoSinMemoria(a-1,b/a,c-1),recursivoSinMemoria(a-2,b/2,c/2));
            }else {
                ac = String.format("(%s/%s)", recursivoSinMemoria(a/2,b-2,c/2),recursivoSinMemoria(a/3,b-1,c/3));
            }
        return ac;
    }

//---------------------------------RECURSIVA CON MEMORIA-------------------------------------------------------------------

    public static String recursivaConMemoria(Integer a, Integer b,Integer c) {
        Map<Tuple, String> m = new HashMap<>();
        return gRecConMemoria(a, b,c, m);
    }

    private static String gRecConMemoria(Integer a, Integer b,Integer c, Map<Tuple, String> m) {
        String ac = null;
        Tuple key = tupla.of(a, b,c);

        if (m.containsKey(key)) {
            ac = m.get(key);
        } else if(a < 2 && b <= 2 || c < 2) {
            ac = String.format("(%d+%d+%d)", a,b,c);
        }else if(a<3 || b<3 && c<3) {
            ac = String.format("(%d-%d-%d)", a,b,c);
            }else if(b%a== 0 && (a%2==0 || b%3 ==0)) {
                ac = String.format("(%s*%s)", recursivoSinMemoria(a-1,b/a,c-1),recursivoSinMemoria(a-2,b/2,c/2),m);
            }else {
                ac = String.format("(%s/%s)", recursivoSinMemoria(a/2,b-2,c/2),recursivoSinMemoria(a/3,b-1,c/3),m);
            }
        return ac;
    }
    public record tupla (Integer a,Integer b,Integer c) {
        public static Tuple of(Integer a, Integer b, Integer c) {
            return new Tuple(a, b, c);
        }
    }

//---------------------------------ITERATIVA-------------------------------------------------------------------
public static String Iterativo(Integer a, Integer b, Integer c ) {
    Map<Tuple, String> m = new HashMap<>();
    String ac = null;

    for (int i = 0; i <= a; i++) {
        for (int j = 0; j <= b; j++) {
            for (int k = 0; k <= c; k++) {
               if(i < 2 && j <= 2 || k < 2) {
                     ac = String.format("(%d+%d+%d)", i,j,k);
                 }else if(i<3 || j<3 && k<3) {
                     ac = String.format("(%d-%d-%d)", i,j,k);
                     }else if(j%i== 0 && (i%2==0 || j%3 ==0)) {
                         ac = String.format("(%s*%s)", m.get(Tuple.of(i-1,j/i,k-1)),m.get(Tuple.of(i-2,j/2,k/2)));
                     }
                     else {
                         ac = String.format("(%s/%s)", m.get(Tuple.of(i/2,j-2,k/2)), m.get(Tuple.of(i/3,j-1,k/3)));
                     }
               m.put(Tuple.of(i, j,k), ac);
            }
        }
    }
    return ac;
}
```

EJERCICIO 4 TEST



```java
public class TestEjercicio4 {

    public static void main(String[] args) {
        String file = "ficheros/testsAlumnos/PI1Ej4DatosEntrada.txt";
        List<Tuple> ls = Files2.streamFromFile(file)
                .map(x → Tuple
                        .parse(x))
                .collect(Collectors.toList());

        System.out.println(
                "*******************************************************************************************
        System.out.println("Entrada: " + ls);
        System.out.println(
                "*******************************************************************************************
                        + "\n");

        for (Tuple ex : ls) {
            System.out.println(
                    "Recursivo SIN MEMORIA " + ex + " ⟹ " + Ejercicio4.recursivoSinMemoria(ex.a, ex.b,ex.c));
            System.out.println(
                    "Recursivo  CON MEMORIA " + ex + "⟹ " + Ejercicio4.recursivaConMemoria(ex.a, ex.b, ex.c));
            System.out.println(
                    "              ITERATIVO " + ex + "⟹ " + Ejercicio4.Iterativo(ex.a, ex.b, ex.c)+"\n");
        }
    }

    public record Tuple(Integer a, Integer b, Integer c) {

        public static Tuple of(Integer a, Integer b, Integer c) {
            return new Tuple(a, b, c);
        }

        public static Tuple parse(String st) {
            List<Integer> par = Arrays.stream(st.split(","))
                    .map(x → Integer.parseInt(x.trim()))
                    .collect(Collectors.toList());
            return of(par.get(0), par.get(1), par.get(2));
        }

        public String toString() {
            return String.format("(%d, %d , %d)", a, b, c);
        }
    }
}
```



```
*************************************************************************************************************************************************************
Entrada: [(30, 20 , 10), (20, 30 , 10), (20, 10 , 30), (20, 15 , 10), (40, 30 , 20), (60, 50 , 40)]
*************************************************************************************************************************************************************

Recursivo SIN MEMORIA (30, 20 , 10) ⟹ ((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
Recursivo  CON MEMORIA (30, 20 , 10)⟹ ((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))
              ITERATIVO (30, 20 , 10)⟹ ((((3+14+1)/(2+15+0))/(5+17+1))/((5+17+1)/(3+18+1)))

Recursivo SIN MEMORIA (20, 30 , 10) ⟹ ((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
Recursivo  CON MEMORIA (20, 30 , 10)⟹ ((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))
              ITERATIVO (20, 30 , 10)⟹ ((((2+24+1)/(1+25+0))/(3+27+1))/((3+27+1)/(2+28+1)))

Recursivo SIN MEMORIA (20, 10 , 30) ⟹ ((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
Recursivo  CON MEMORIA (20, 10 , 30)⟹ ((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))
              ITERATIVO (20, 10 , 30)⟹ ((((2-4-3)/(1-5-2))/((1-5-2)/(1+6+1)))/(((1-5-2)/(1+6+1))/(2-8-3)))

Recursivo SIN MEMORIA (20, 15 , 10) ⟹ ((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
Recursivo  CON MEMORIA (20, 15 , 10)⟹ ((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))
              ITERATIVO (20, 15 , 10)⟹ ((((2+9+1)/(1+10+0))/(3+12+1))/((3+12+1)/(2+13+1)))

Recursivo SIN MEMORIA (40, 30 , 20) ⟹ (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))
Recursivo  CON MEMORIA (40, 30 , 20)⟹ (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))
              ITERATIVO (40, 30 , 20)⟹ (((((2+22+1)/(1+23+0))/(3+25+1))/((3+25+1)/(2+26+1)))/(((3+25+1)/(2+26+1))/((3+7+1)*(2+14+1))))

Recursivo SIN MEMORIA (60, 50 , 40) ⟹ ((((((2+14+1)*(1+21+1))/(2+43+1))/(((2+7+1)/(1+8+0))*(3+22+1)))/((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3
Recursivo  CON MEMORIA (60, 50 , 40)⟹ ((((((2+14+1)*(1+21+1))/(2+43+1))/(((2+7+1)/(1+8+0))*(3+22+1)))/((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3
              ITERATIVO (60, 50 , 40)⟹ ((((((2+14+1)*(1+21+1))/(2+43+1))/(((2+7+1)/(1+8+0))*(3+22+1)))/((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0))))/(((((2+7+1)/(1+8+0))*(3+22+1))/((1+44+1)/(1+45+0)))/(((2+6+1)/(1+7+1))*((3
```