Práctica 1 - PREDA
Devolución de cambio
Programación Dinámica

Nombre: Pablo Suárez Peña

DNI: 53647180W

Centro Asociado: Asturias

Email: psuarez142@alumno.uned.es

Código de Asignatura: 71902019

0) INDICE:

1) INTRODUCCIÓN A LA PRÁCTICA:

a. Sistema Operativo : macOS Monterrey 12.6.8
b. IDE : BlueJ 5.2.0
c. Versión de Java 17.0.4.1

2) INTRODUCCIÓN TEÓRICA:

Como podemos estudiar en el apartado 5.3 del libro de texto y en este extracto extraído del mismo podemos ver como resolver este problema con el esquema de programación dinámica.

En primer lugar debemos pensar como plantear el problema de forma incremental. Consideramos el tipo de moneda de mayor valor, *XN.*

Si *XN* > C entonces la descartamos y pasamos a considerar monedas de menor valor.

Si *XN* :S C tenemos dos opciones: o tomar una moneda de tipo *XN,* Y completar la cantidad restante C - *XN* con otras monedas, o no tomar ninguna moneda de tipo *XN* y completar la cantidad C con monedas de menor valor.

De las dos opciones nos quedamos con la que requiera un número menor de monedas.

El problema lo podemos expresar de la siguiente forma cuando consideramos *N* tipos de monedas:

$$cambio(N,C) = \begin{cases} cambio(N-1,C) & \text{si } x_N > C \\ min\{cambio(N-1,C), cambio(N,C-x_N)+1\} & \text{si } x_N \leq C \end{cases}$$

Podemos razonar análogamente para monedas de valores *k* menores que *N* y para cantidades C' menores que C:

$$cambio(k,C') = \begin{cases} cambio(k-1,C') & \text{si } x_k > C' \\ min\{cambio(k-1,C'), cambio(k,C'-x_k)+1\} & \text{si } x_k \leq C' \end{cases}$$

Llegamos a los casos base de la recurrencia cuando completamos la cantidad C:

$$cambio(k,0) = 0 \ \text{ si } 0 \leq k \leq n,$$

o cuando ya no quedan más tipos de monedas por considerar, pero aún no se ha completado la cantidad C:

$$cambio(0,C') = \infty \text{ si } 0 < C' \leq C.$$

Podemos construir una tabla para almacenar los resultados parciales que tenga una fila para cada tipo de moneda y una columna para cada cantidad posible entre 1 y C. Cada posición *t[N,C],* será el número mínimo de monedas necesario para dar una cantidad }, con O:S } :S C, utilizando sólo monedas de los tipos entre 1 e *i,* con O:S *i* :S *N.* La solución al problema será por tanto el contenido de la casilla *t[N,C].* Palia construir la tabla empezamos rellenando los casos base *t[i,O]* = O, para todo *i* con O:S *i* :S *N.* A continuación podemos rellenar la tabla bien por filas de izquierda a derecha, o bien por columnas de arriba a abajo.

El algoritmo para el problema puede tomar la siguiente forma:

```
tipo Tabla = matriz[1..N,0..C] de entero
tipo Vector = matriz[0..N] de entero
fun DarCambio(C: entero, moneda: Vector): Tabla
    var
        t: Tabla
        i,j: entero
    fvar
    para i ← 1 hasta N hacer
        t[i,0] ← 0
    fpara
    para j ← 1 hasta C hacer
        para i ← 1 hasta N hacer
            si i = 1 ∧ moneda[i] > j entonces
                t[i,j] ← ∞
            sino
                si i = 1 entonces
                    t[i,j] ← 1 + t[1,j-moneda[i]]
                sino
                    si j < moneda[i] entonces
                        t[i,j] ← t[i-1,j]
                    sino
                        t[i,j] ← min(t[i-1,j], t[i,j-moneda[i]] + 1)
                    fsi
                fsi
            fsi
        fpara
    fpara
    dev t
ffun
```

El algoritmo comienza inicializando la primera columna de la tabla a O. Después va completando las restantes columnas hasta llegar a la C. Para cada columna empieza por comprobar si el valor de la moneda más pequeña es mayor que la cantidad a completar. Si es así se coloca una marca en la tabla 0 0 para indicar la imposibilidad de hacer la operación. Si no es así, se asigna la casilla correspondiente a la moneda más pequeña. Si no estamos en la primera de la filas se comprueba si la cantidad a completar *j* es menor que la moneda asociada a esa fila.

Si no es así no podemos utilizar esa moneda y por tanto, el valor de esa casilla no cambia respecto del de la fila anterior. Si es menor, entonces podemos encontrar nuevas formas de completar la cantidad *j,* de las que seleccionamos la mejor.
Para estudiar el coste del algoritmo calculamos el tamaño de la tabla que hay que cons- truir: *N* x (C+1). Como las operaciones aritméticas involucradas son de coste constante, el tiempo de ejecución está en *e(NC).*

Si lo que queremos es que el algoritmo no sólo nos diga cuál es la cantidad mínima de monedas, sino también qué monedas hay que devolver tenemos dos alternativas. Una de ellas es ir almacenando en la tabla no sólo el número de monedas, sino también los tipos de las monedas. Esto llevaría a un aumento importante del coste de memoria. La mejor alternativa es utilizar la tabla construida para saber el número de monedas. Partimos de la casilla final. Para cada casilla t[i,j] el algoritmo va comprobando si su valor ha variado respecto a la casilla de la fila superior. Si no ha variado podemos deducir que no se ha empleado ninguna moneda del tipo de la fila *i,* y pasamos a comprobar la casilla superior t[i-l,j]. Si ha variado, anotamos que se ha utilizado una moneda de ese tipo *Xi* y nos movemos a la casilla t[i, j-moneda[i]], para ver qué monedas se han utilizado para dar la cantidad restante. Siguiendo esta estrategia terminaremos alcanzando la casilla t[O,O] en la que ya no queda ninguna cantidad pendiente. El algoritmo puede tomar la forma que se muestra a continuación:

```
tipo Tabla = matriz[1..N,0..C] de entero
tipo Vector = matriz[0..N] de entero
fun seleccionar_monedas(C: entero, moneda: Vector, t:Tabla, seleccion: Vector)
    var
        i,j: entero
    fvar
    para i ← 0 hasta N hacer
        seleccion[i] ← 0
    fpara
    i ← N
    j ← C
    mientras j > 0 hacer
        si i > 1 ∧ t[i,j] = t[i-1,j] entonces
            i ← i - 1
        sino
            seleccion[i] ← seleccion[i] + 1
            j ← j - moneda[i]
        fsi
    fmientras
ffun
```

3) CUESTIONES TEÓRICAS:

1. *Indica y razona sobre el coste temporal y espacial del algoritmo.*

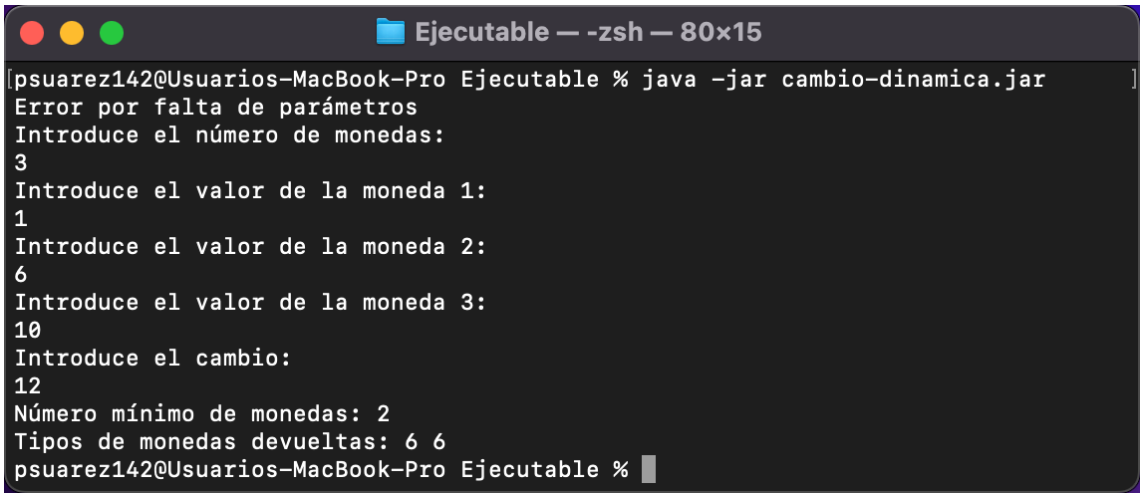
   Para estudiar el coste del algoritmo calculamos el tamaño de la tabla que hay que construir: *N* x (C+1). Como las operaciones aritméticas involucradas son de coste constante, el tiempo de ejecución está en *e(NC).*

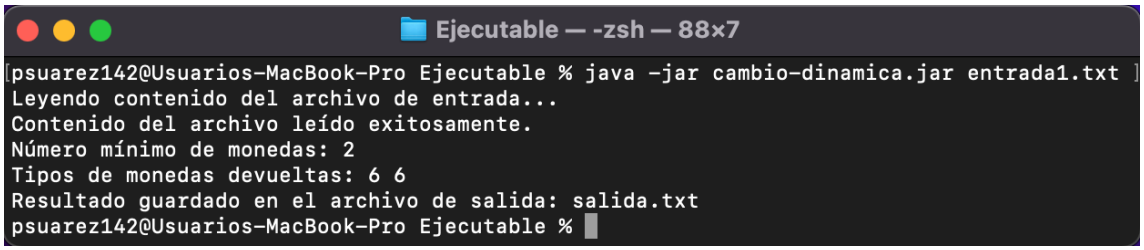
2. *Explica qué otros esquemas pueden resolver el problema y razona sobre su idoneidad.*

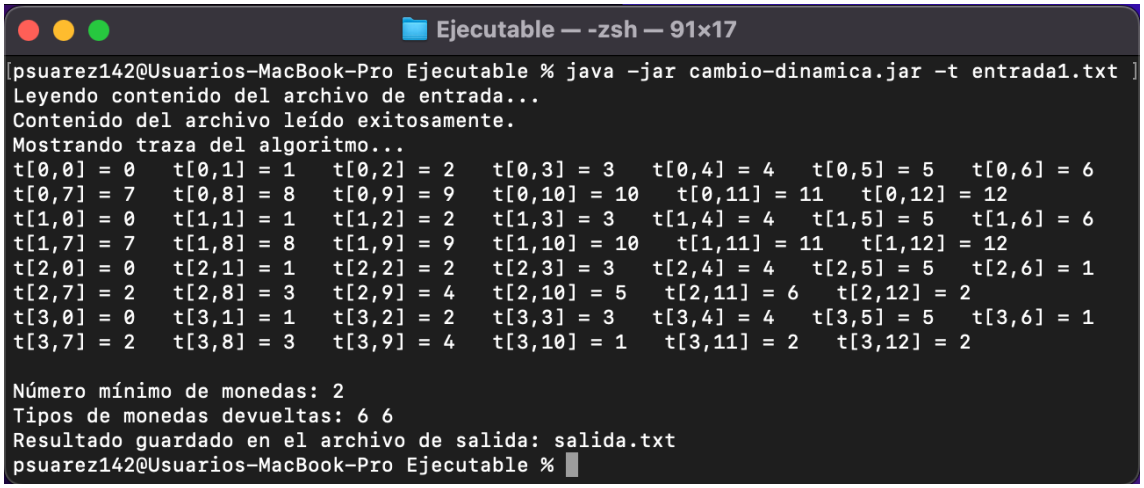
   Recordemos la formulación del problema de la devolución de cambio planteado en el capítulo 3, dedicado a los algoritmos voraces. Se tiene un conjunto de *N* tipos de monedas, cada una con un valor *Xi.* Se supone que contamos con una cantidad ilimitada de monedas de cada tipo. El problema consiste en hallar el número mínimo de monedas que necesitamos para dar una cierta cantidad C.

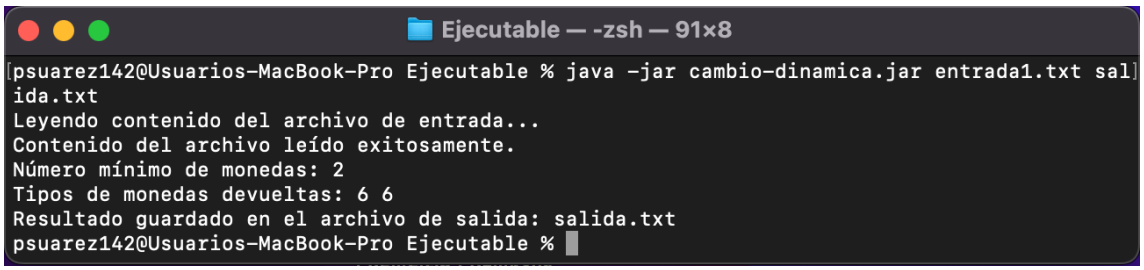
   Un algoritmo voraz que eligiese siempre la moneda de mayor valor que se pueda tomar para acercarse a la cantidad C no funcionaría para cualquier conjunto de tipos de monedas. Por ejemplo, si tenemos monedas de valores 1, 6 y 10, y la cantidad a completar fuera 24, la estrategia voraz elegiría las siguientes monedas: 10, 10, 1, 1, 1 Y 1. Es decir se darían 6 monedas, cuando es posible devolver esa cantidad con sólo 4 monedas de valor 6.

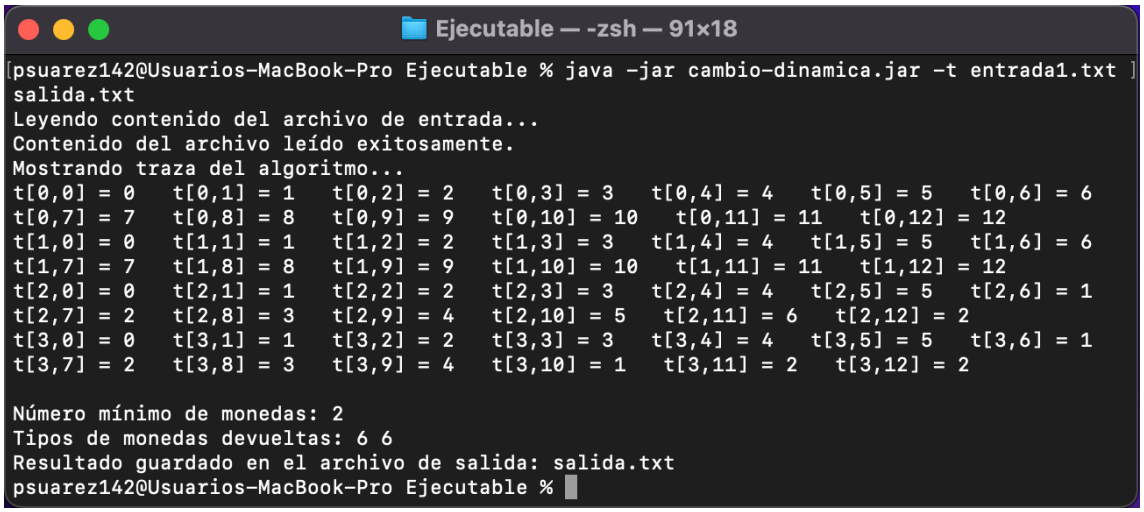
4) EJEMPLO DE EJECUCIÓN:

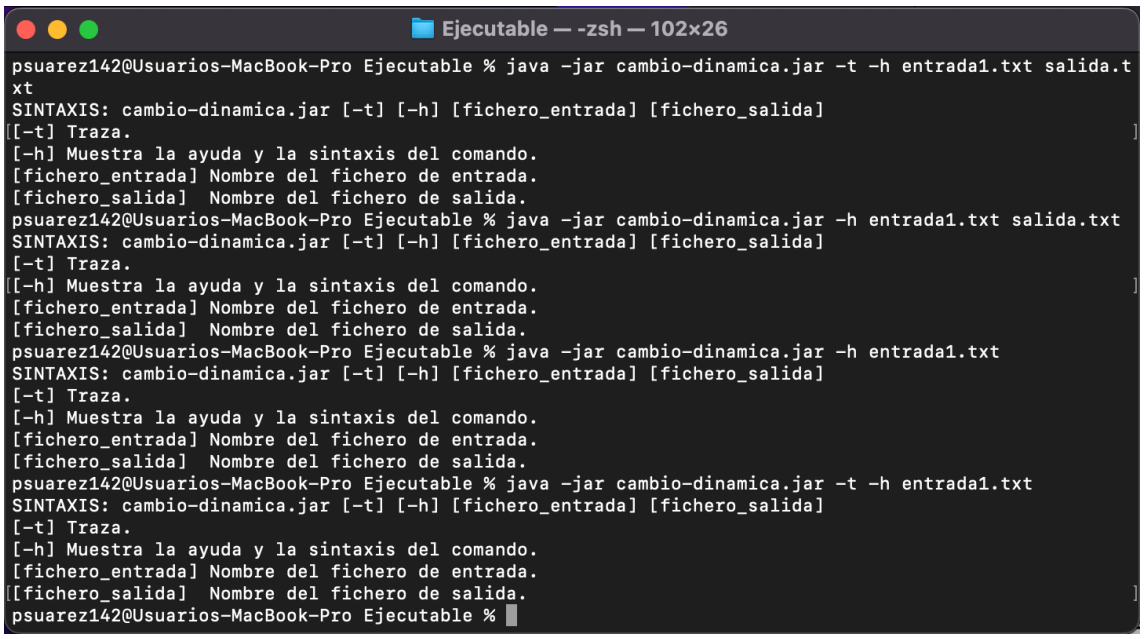






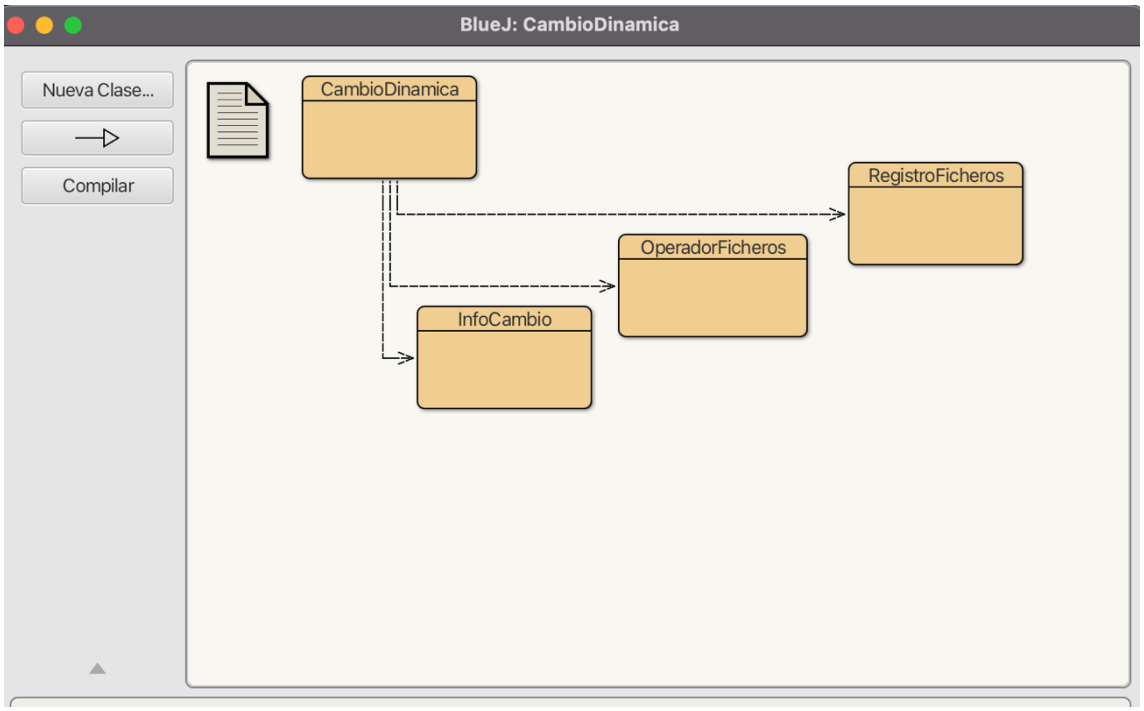



Ejecutable — -zsh — 91×18
psuarez142@Usuarios-MacBook-Pro Ejecutable % java -jar cambio-dinamica.jar -t entrada1.txt salida.txt
Leyendo contenido del archivo de entrada...
Contenido del archivo leído exitosamente.
Mostrando traza del algoritmo...
t[0,0] = 0 t[0,1] = 1 t[0,2] = 2 t[0,3] = 3 t[0,4] = 4 t[0,5] = 5 t[0,6] = 6
t[0,7] = 7 t[0,8] = 8 t[0,9] = 9 t[0,10] = 10 t[0,11] = 11 t[0,12] = 12
t[1,0] = 0 t[1,1] = 1 t[1,2] = 2 t[1,3] = 3 t[1,4] = 4 t[1,5] = 5 t[1,6] = 6
t[1,7] = 7 t[1,8] = 8 t[1,9] = 9 t[1,10] = 10 t[1,11] = 11 t[1,12] = 12
t[2,0] = 0 t[2,1] = 1 t[2,2] = 2 t[2,3] = 3 t[2,4] = 4 t[2,5] = 5 t[2,6] = 1
t[2,7] = 2 t[2,8] = 3 t[2,9] = 4 t[2,10] = 5 t[2,11] = 6 t[2,12] = 2
t[3,0] = 0 t[3,1] = 1 t[3,2] = 2 t[3,3] = 3 t[3,4] = 4 t[3,5] = 5 t[3,6] = 1
t[3,7] = 2 t[3,8] = 3 t[3,9] = 4 t[3,10] = 1 t[3,11] = 2 t[3,12] = 2
Número mínimo de monedas: 2
Tipos de monedas devueltas: 6 6
Resultado guardado en el archivo de salida: salida.txt
psuarez142@Usuarios-MacBook-Pro Ejecutable %


Ejecutable — -zsh — 102×26
psuarez142@Usuarios-MacBook-Pro Ejecutable % java -jar cambio-dinamica.jar -t -h entrada1.txt salida.txt
SINTAXIS: cambio-dinamica.jar [-t] [-h] [fichero_entrada] [fichero_salida]
[-t] Traza.
[-h] Muestra la ayuda y la sintaxis del comando.
[fichero_entrada] Nombre del fichero de entrada.
[fichero_salida] Nombre del fichero de salida.
psuarez142@Usuarios-MacBook-Pro Ejecutable % java -jar cambio-dinamica.jar -h entrada1.txt salida.txt
SINTAXIS: cambio-dinamica.jar [-t] [-h] [fichero_entrada] [fichero_salida]
[-t] Traza.
[-h] Muestra la ayuda y la sintaxis del comando.
[fichero_entrada] Nombre del fichero de entrada.
[fichero_salida] Nombre del fichero de salida.
psuarez142@Usuarios-MacBook-Pro Ejecutable % java -jar cambio-dinamica.jar -h entrada1.txt
SINTAXIS: cambio-dinamica.jar [-t] [-h] [fichero_entrada] [fichero_salida]
[-t] Traza.
[-h] Muestra la ayuda y la sintaxis del comando.
[fichero_entrada] Nombre del fichero de entrada.
[fichero_salida] Nombre del fichero de salida.
psuarez142@Usuarios-MacBook-Pro Ejecutable % java -jar cambio-dinamica.jar -t -h entrada1.txt
SINTAXIS: cambio-dinamica.jar [-t] [-h] [fichero_entrada] [fichero_salida]
[-t] Traza.
[-h] Muestra la ayuda y la sintaxis del comando.
[fichero_entrada] Nombre del fichero de entrada.
[fichero_salida] Nombre del fichero de salida.
psuarez142@Usuarios-MacBook-Pro Ejecutable %

5) CÓDIGO FUENTE:

5.a) Diagrama de clases:



Captura de pantalla de BlueJ.

5.b) Código Fuente de Completo*:

En nuestro código fuente hemos implementado 4 clases:

- Clase CambioDinámica
- Clase InfoCambio
- Clase OperadorFicheros
- Clase RegistroFicheros

CambioDinamica:

Esta clase representa la lógica principal del programa. Aquí se procesan los argumentos de entrada, se lee el contenido de un fichero, se realiza la inicialización de variables y se llama a métodos de la clase InfoCambio para calcular el cambio necesario y obtener información sobre las monedas. Finalmente, se guarda el resultado en un fichero de salida o se imprime en la consola.

```
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class CambioDinamica {

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Error por falta de parámetros");
            solicitarEntradaUsuario();
        } else {
            ejecutarCaso(args);
        }
    }

  private static void ejecutarCaso(String[] args) {
    boolean traza = false;
    String ficheroEntrada = null;
    String ficheroSalida = null;
    boolean mostrarAyuda = false;

    for (int i = 0; i < args.length; i++) {
        switch (args[i]) {
            case "-t":
                traza = true;
                break;
            case "-h":
                validarOpcionAyuda(args);
                return; // Terminar la ejecución después de
mostrar ayuda
            default:
                 if ("-h".equals(args[i])) {
                    mostrarAyuda = true;
                } else if (ficheroEntrada == null) {
                    ficheroEntrada = args[i];
                } else if (ficheroSalida == null) {
                    ficheroSalida = args[i];
                } else {
```

```
                    manejarErrorArgumentos();
                    return;
                }
                break;
            }
        }

          if (mostrarAyuda) {
            validarOpcionAyuda(args);
            return;
        }

        if (ficheroEntrada == null) {
            manejarErrorArgumentos();
            return;
        }

        if (ficheroSalida == null) {
            ficheroSalida = "salida.txt";
        }

        validarFichero(ficheroEntrada, ficheroSalida, traza);
    }

        private static void manejarErrorArgumentos() {
            System.out.println("SINTAXIS: cambio-dinamica.jar [-t]
    [-h] [fichero_entrada] [fichero_salida]");
            System.out.println("[-t] Traza.");
            System.out.println("[-h] Muestra la ayuda y la sintaxis
    del comando.");
            System.out.println("[fichero_entrada] Nombre del fichero
    de entrada.");
            System.out.println("[fichero_salida]  Nombre del fichero
    de salida.");
            System.exit(1);
        }

        private static void ejecutarLogica(String ficheroEntrada,
    String ficheroSalida, boolean traza) {
            try {
                System.out.println("Leyendo contenido del archivo de
    entrada...");
                List<String> contenidoFichero =
    OperadorFicheros.leerFichero(ficheroEntrada);
                System.out.println("Contenido del archivo leído
    exitosamente.");

                if (contenidoFichero.size() != 3) {
```

```
                System.out.println("Error en el formato del
fichero de entrada. Verifica los argumentos.");
                solicitarEntradaUsuario();
            }

            int n = Integer.parseInt(contenidoFichero.get(0));

            if (n <= 0) {
                System.out.println("Error: El número de monedas
debe ser un entero mayor que 0.");
                solicitarEntradaUsuario();
            }

            String[] monedasString =
contenidoFichero.get(1).split(" ");

            if (monedasString.length != n) {
                System.out.println("Error: El número de tipos de
monedas no coincide con el número de monedas especificado.");
                solicitarEntradaUsuario();
            }

            int[] monedas = new int[n];

            for (int i = 0; i < n; i++) {
                monedas[i] = Integer.parseInt(monedasString[i]);

                if (monedas[i] <= 0 || (i > 0 && monedas[i] <=
monedas[i - 1])) {
                    System.out.println("Error: El valor de las
monedas debe ser mayor que 0 y consecutivo estrictamente
creciente.");
                    solicitarEntradaUsuario();
                }
            }

            int cantidadDevolver =
Integer.parseInt(contenidoFichero.get(2));

            if (cantidadDevolver < 0 || cantidadDevolver <
monedas[0]) {
                System.out.println("Error: La cantidad a
devolver debe ser mayor o igual a alguno de los valores del tipo
de cambio.");
                solicitarEntradaUsuario();
            }

            InfoCambio infoCambio = new
InfoCambio(cantidadDevolver, n, monedas, traza);
```

```
            infoCambio.getCambio(traza);

            if (traza) {
                System.out.println("Mostrando traza del
algoritmo...");

System.out.println(infoCambio.getTrazaSolucion());
            }

            System.out.println("Número mínimo de monedas: " +
infoCambio.getCambioSolucion());
            System.out.println("Tipos de monedas devueltas: " +
infoCambio.getTipoMonedas());

            if (ficheroSalida != null) {
                String contenidoGuardar1 =
infoCambio.getCambioSolucion();
                String contenidoGuardar2 =
infoCambio.getTipoMonedas();
                String contenidoGuardar = contenidoGuardar1 +
System.getProperty("line.separator") +
                        contenidoGuardar2;
                OperadorFicheros.guardarFichero(ficheroSalida,
contenidoGuardar);
                System.out.println("Resultado guardado en el
archivo de salida: " + ficheroSalida);
            }
        } catch (NumberFormatException |
IndexOutOfBoundsException e) {
            System.out.println("Error en el formato del fichero
de entrada. Verifica los argumentos.");
            solicitarEntradaUsuario();
        } catch (IOException e) {
            System.out.println("Error al leer el fichero de
entrada.");
            e.printStackTrace();
        }
    }
    private static void validarFichero(String
rutaFicheroEntrada, String rutaFicheroSalida, boolean traza) {
    File ficheroEntrada = new File(rutaFicheroEntrada);

    if (!ficheroEntrada.exists() || !ficheroEntrada.isFile()) {
        // Si es la opción -t, simplemente indicamos que la
traza está activada y continuamos
        if (traza) {
            ejecutarLogica(rutaFicheroEntrada,
rutaFicheroSalida, traza);
            return;
```

```
        }

        System.out.println("Error: El fichero de entrada no
existe o no es válido.");
        System.exit(1);
    }

    // Si el fichero de salida es null, se utiliza la ruta del
fichero de entrada con un nuevo nombre
    String ficheroSalida = (rutaFicheroSalida != null) ?
rutaFicheroSalida : "salida.txt";

    // Llamar a ejecutarLogica con las rutas válidas
    ejecutarLogica(ficheroEntrada.getAbsolutePath(),
ficheroSalida, traza);
    }

     private static void validarOpcionAyuda(String[] args) {
        if (args.length == 2 || args.length == 4) {
            ejecutarAyuda();
        } else {
            manejarErrorArgumentos();
        }
    }

    private static void ejecutarAyuda() {
        System.out.println("SINTAXIS: cambio-dinamica.jar [-t]
[-h] [fichero_entrada] [fichero_salida]");
        System.out.println("[-t] Traza.");
        System.out.println("[-h] Muestra la ayuda y la sintaxis
del comando.");
        System.out.println("[fichero_entrada] Nombre del fichero
de entrada.");
        System.out.println("[fichero_salida]  Nombre del fichero
de salida.");
        System.exit(0);
    }

    private static void solicitarEntradaUsuario() {
        Scanner scanner = new Scanner(System.in);

        try {
            System.out.println("Introduce el número de
monedas:");
            int n = scanner.nextInt();

            if (n <= 0) {
                throw new IllegalArgumentException("Error: El
número de monedas debe ser un entero mayor que 0.");
```

```
            }

            int[] monedas = new int[n];
            for (int i = 0; i < n; i++) {
                System.out.println("Introduce el valor de la
moneda " + (i + 1) + ":");
                monedas[i] = scanner.nextInt();

                if (monedas[i] <= 0 || (i > 0 && monedas[i] <=
monedas[i - 1])) {
                    throw new IllegalArgumentException("Error:
El valor de las monedas debe ser mayor que 0 y consecutivo
estrictamente creciente.");
                }
            }

            System.out.println("Introduce el cambio:");
            int cantidadDevolver = scanner.nextInt();

            if (cantidadDevolver < 0 || cantidadDevolver <
monedas[0]) {
                throw new IllegalArgumentException("Error: La
cantidad a devolver debe ser mayor o igual a alguno de los
valores del tipo de cambio.");
            }

            InfoCambio infoCambio = new
InfoCambio(cantidadDevolver, n, monedas, true);
            infoCambio.getCambio(true); // Mostrar traza

            System.out.println("Número mínimo de monedas: " +
infoCambio.getCambioSolucion());
            System.out.println("Tipos de monedas devueltas: " +
infoCambio.getTipoMonedas());

        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
            solicitarEntradaUsuario(); // Si hay un error,
solicitar la entrada nuevamente
        } finally {
            scanner.close();
        }
    }
   }
```

InfoCambio:

La clase InfoCambio implementa el algoritmo de programación dinámica para calcular el número mínimo de monedas necesarias para devolver una cantidad específica de cambio. Almacena información sobre la cantidad a devolver, la cantidad de monedas disponibles y los tipos de monedas. Además, proporciona métodos para obtener la solución del cambio y el tipo de monedas devueltas.

```
public class InfoCambio {
    private int cantidadDevolver;
    private int cantidadMonedasDif;
    private int[] tiposMonedas;
    private int[][] tabla;
    private int[] monedasDevueltas;
    private int numMinMonedasDevueltas;
    private String tipoMonedasDevueltas;
    private String trazaSolucion;
    private boolean mostrarTraza;
    private int n;

    public InfoCambio(int cantidadDevolver, int cantidadMonedas,
int[] tiposMonedas, boolean mostrarTraza) {
        this.cantidadDevolver = cantidadDevolver;
        this.cantidadMonedasDif = cantidadMonedas;
        this.tiposMonedas = tiposMonedas;
        this.mostrarTraza = mostrarTraza;
        this.tabla = new int[tiposMonedas.length +
1][cantidadDevolver + 1];
        this.monedasDevueltas = new int[tiposMonedas.length];
        this.trazaSolucion = "";
        this.n = cantidadMonedas;
    }

   public void getCambio(Boolean traza) {
        if (mostrarTraza) {
            trazaSolucion += "La cantidad a devolver es: " +
cantidadDevolver + System.getProperty("line.separator");
            trazaSolucion += "con " + cantidadMonedasDif + "
tipos diferentes de monedas" +
System.getProperty("line.separator");
        }

        for (int i = 0; i <= n; i++) {  // Ajustamos la
condición de iteración a n
            for (int j = 0; j <= cantidadDevolver; j++) {
                if (traza) {
                    trazaSolucion += tabla[i][j] + " ";
```

```
                    if (j == cantidadDevolver) {
                        trazaSolucion +=
System.getProperty("line.separator");
                    }
                }

                if (i == 0) {
                    tabla[i][j] = j;
                } else if (tiposMonedas[i - 1] > j) {
                    tabla[i][j] = tabla[i - 1][j];
                } else {
                    tabla[i][j] = Math.min(tabla[i - 1][j],
tabla[i][j - tiposMonedas[i - 1]] + 1);
                }
            }
        }

        numMinMonedasDevueltas = tabla[n][cantidadDevolver];  //
Ajustamos la obtención del resultado
    }


    public String getTipoMonedas() {
    int cantidadRestante = cantidadDevolver;
    int i = tiposMonedas.length;
    int j = cantidadDevolver;
    tipoMonedasDevueltas = "";

    while (i > 0) {
        if (tabla[i][j] == tabla[i - 1][j]) {
            i = i - 1;
        } else {
            tipoMonedasDevueltas += tiposMonedas[i - 1] + " ";
            j = j - tiposMonedas[i - 1];
        }
    }
    return tipoMonedasDevueltas.trim();
    }


    public String getCambioSolucion() {
        return String.valueOf(numMinMonedasDevueltas);
    }

  public String getTrazaSolucion() {
        if (mostrarTraza) {
            StringBuilder trazaSolucion = new StringBuilder();

            for (int i = 0; i <= tiposMonedas.length; i++) {
```

```
                for (int j = 0; j <= cantidadDevolver; j++) {
                    trazaSolucion.append(String.format("t[%d,%d]
= %d   ", i, j, tabla[i][j]));
                }
                trazaSolucion.append(System.lineSeparator());
            }

            return trazaSolucion.toString();
        } else {
            return "La opción de traza no está habilitada.";
        }
    }
    private void rellenarTraza() {
        for (int i = 0; i <= tiposMonedas.length; i++) {
            for (int j = 1; j <= cantidadDevolver; j++) {
                trazaSolucion += " " + tabla[i][j];
                if (j == cantidadDevolver) {
                    trazaSolucion +=
System.getProperty("line.separator");
                }
            }
        }
        trazaSolucion += System.getProperty("line.separator");
    }

        // Método getter para mostrarTraza
    public boolean isMostrarTraza() {
        return mostrarTraza;
    }
    }
```

RegistroFicheros:

La clase RegistroFicheros se encarga de analizar los argumentos de entrada y gestionar la información relacionada con los ficheros de entrada y salida. También contiene métodos para obtener la traza, la ayuda y los nombres de los ficheros. Un método importante es getCantidadDevolver(), que debe ser implementado para obtener la cantidad a devolver.

```
import java.io.IOException;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.Arrays;

class RegistroFicheros {
    private String[] args;
    private Boolean traza;
    private Boolean ayuda;
    private String ficheroEntrada;
    private String ficheroSalida;
    private int cantidadDevolver;

    public RegistroFicheros(String[] args) {
        this.args = args != null ? args : new String[0];
        this.traza = false;
        this.ayuda = false;
        this.ficheroEntrada = null;
        this.ficheroSalida = null;
    }

    public Boolean esOrdenCorrecto() {
        int idxT = obtenerIndiceArgumento("-t");
        int idxH = obtenerIndiceArgumento("-h");
        int idxEntrada = obtenerIndiceFicheroEntrada();
        int idxSalida = obtenerIndiceFicheroSalida();

        return (idxT <= idxH && idxH <= idxEntrada && idxEntrada
<= idxSalida) ||
                (idxH <= idxT && idxT <= idxEntrada &&
idxEntrada <= idxSalida);
    }

    private int obtenerIndiceArgumento(String opcion) {
    for (int i = 0; i < this.args.length; i++) {
        if (this.args[i] != null && this.args[i].equals(opcion))
{
            return i;
```

```
        }
    }
    return -1; // Si no se encuentra la opción, devolver -1 o
manejar de acuerdo a tu lógica.
    }

    private int obtenerIndiceFicheroEntrada() {
        return args.length > 0 ? 0 : args.length;
    }

    private int obtenerIndiceFicheroSalida() {
        return args.length > 1 ? 1 : args.length;
    }

    public void anotarArgumentos() {
        Set<String> opciones = new
HashSet<>(Arrays.asList(args));

        traza = opciones.contains("-t");
        ayuda = opciones.contains("-h");

        if (esOrdenCorrecto()) {
            if (args.length > 0) {
                ficheroEntrada =
args[obtenerIndiceFicheroEntrada()];
            }

            if (args.length > 1) {
                ficheroSalida =
args[obtenerIndiceFicheroSalida()];
            }
        } else {
            errorArgumentos();
        }
    }

    public Boolean getMostrarAyuda() {
        return ayuda;
    }

    public void mostrarAyuda() {
        System.out.println("SINTAXIS: cambio-dinamica [-t] [-h]
[fichero entrada] [fichero salida]");
        System.out.println("-t Traza el algoritmo");
        System.out.println("-h Muestra esta ayuda");
        System.out.println("[fichero_entrada] Nombre del fichero
de entrada");
        System.out.println("[fichero_salida]  Nombre del fichero
de salida");
```

```
    }

    public Boolean getTraza() {
        return traza;
    }

    public Boolean getAyuda() {
        return ayuda;
    }

    public String getFicheroEntrada() {
        return ficheroEntrada;
    }

    public String getFicheroSalida() {
        return ficheroSalida;
    }

    public int getCantidadDevolver() {
        return cantidadDevolver;
    }

    public void errorArgumentos() {
        System.out.println("Argumentos erróneos");
        this.mostrarAyuda();
        System.exit(0);
    }
}
```

OperadorFicheros:

La clase OperadorFicheros ofrece métodos estáticos para leer y escribir en ficheros. El método leerFichero() lee el contenido de un fichero de entrada y lo devuelve como una lista de cadenas. El método guardarFichero() guarda un contenido en un fichero de salida. Ambos métodos manejan excepciones de lectura y escritura de ficheros.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class OperadorFicheros {
    public static List<String> leerFichero(String nombreFichero)
throws IOException {
        List<String> lineas = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new
FileReader(nombreFichero))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                lineas.add(linea);
            }
        }
        return lineas;
    }

    public static void guardarFichero(String nombreFichero,
String contenido) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(nombreFichero))) {
            writer.write(contenido);
        }
    }
}
```