**BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA**

BUAP

** FACULTAD DE CIENCIAS DE LA COMPUTACIÓN**

**INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN**

**PROGRAMACIÓN 2**

**“PRACTICA 16. REPORTE PROGRAMAS ARCHIVOS”**

**PROFESOR:**

**DRA. CARMEN CERÓN GARNICA.**

**EQUIPO:  
BRENDA LIZETH COCA GARCIA**

**MATRICULA:201917154**

**BRENDA AILED RODRIGUEZ COLIS**

**MATRICULA:201910848**

**MIGUEL CARREON VAZQUEZ**

**MATRICULA: 201915389**

**IRVYN XICALE CARRERA**

**MATRICULA: 201963582**

**INTRODUCCIÓN**La recursividad es una técnica potente de programación que puede utilizarse en lugar de la iteración para resolver determinados tipos de problemas.

Un método es recursivo cuando entre sus instrucciones se encuentra una llamada a sí mismo. La solución iterativa es fácil de entender.

En esta practica ponemos a prueba los conocimientos adquiridos en recursividad para crear un programa con métodos recursivos.

**PROGRAMA 1**

**DIAGRAMA UML**

****

**CLASE TEXTOS**

//Fecha:3-Mayo-2021  
//Objetivo: PRACTICA 16 ARCHIVOS  
//Autor: Brenda Coca Garcia

import java.io.\*;

public class TextoS {

public static void main(String[] args) {

InputStream entrada = null;

OutputStream salida = null;

try{

entrada = new FileInputStream("origen.txt");

salida = new FileOutputStream("destino.txt");

int c;

while((c=entrada.read()) != -1){

salida.write(c);

}

}catch(FileNotFoundException e){

System.out.println("fichero no encontrado");

}catch(IOException e){

System.out.println("Error de E/S");

}

if(entrada != null) try{

salida.close();

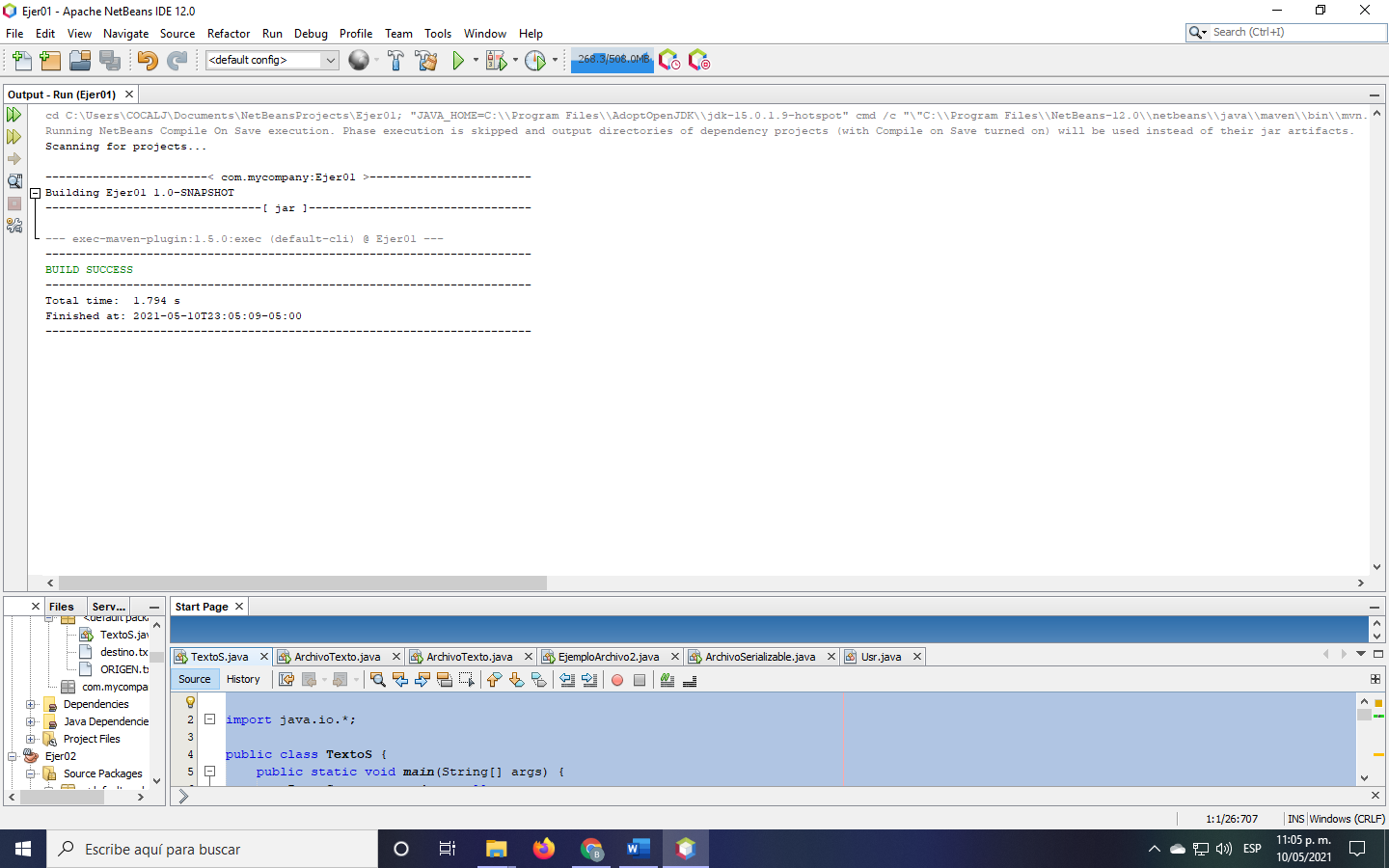
}catch(IOException e){}

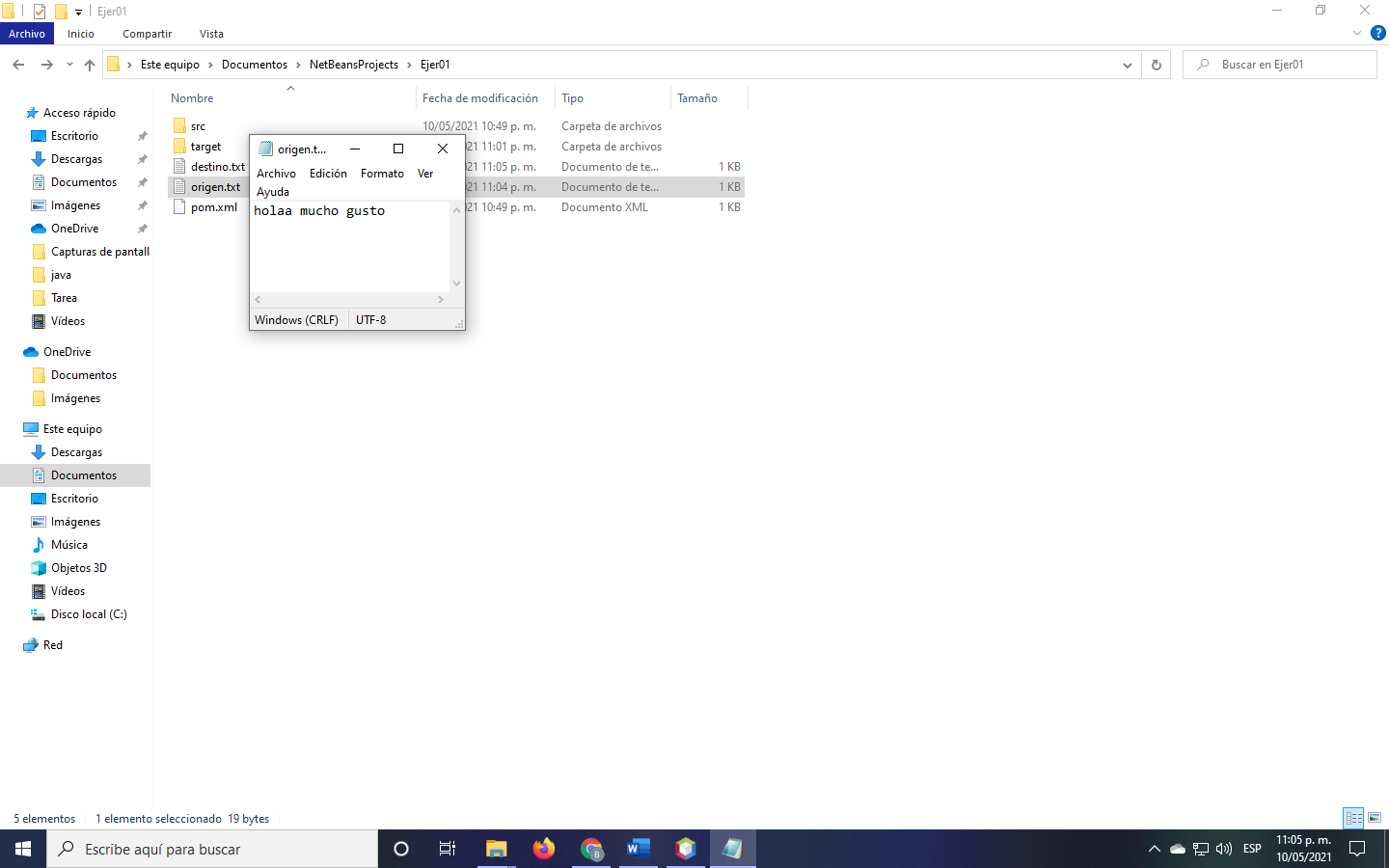
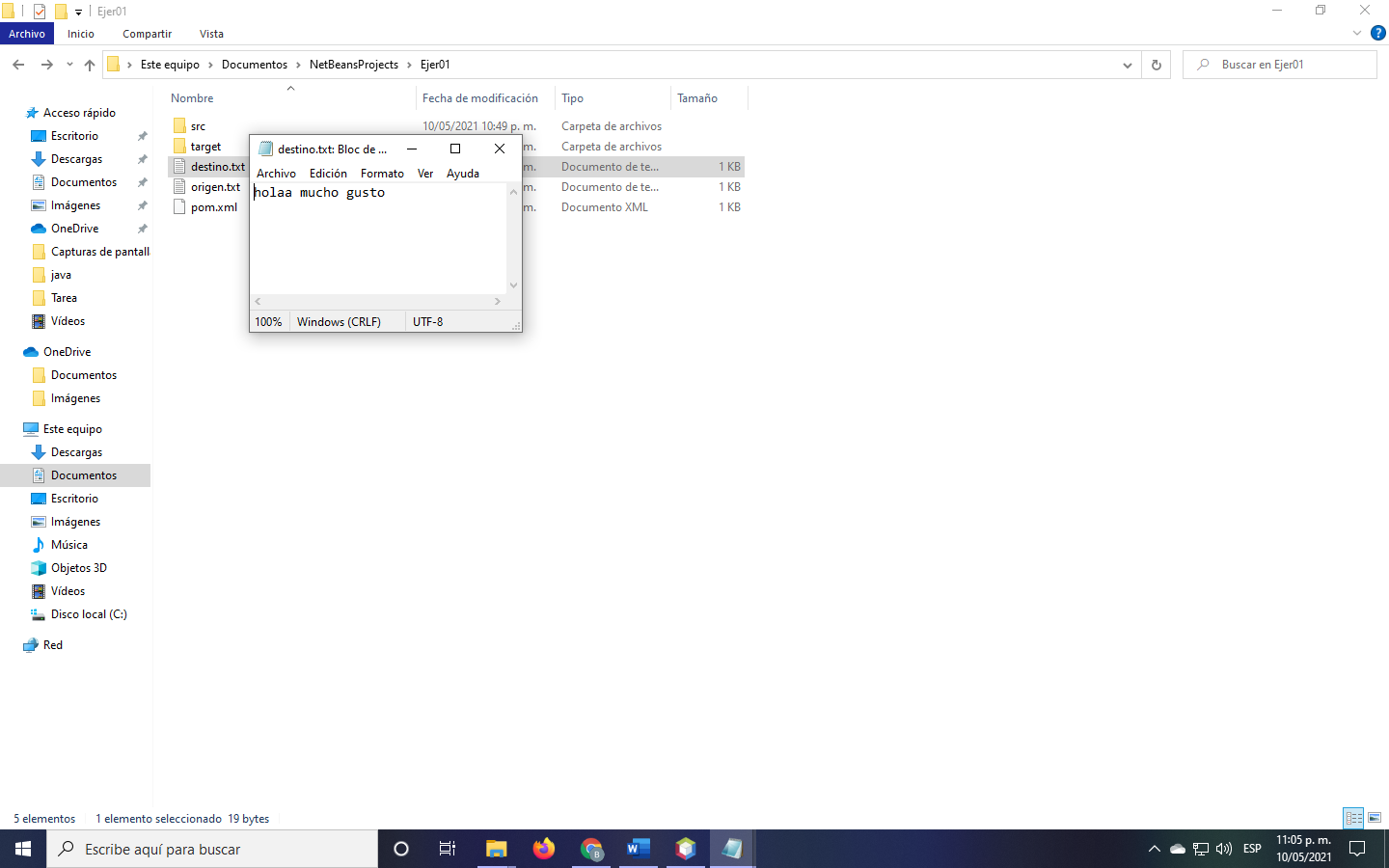
}

}



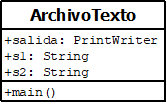
**CORRIDA PROGRAMA**





**PROGRAMA 2**

**DIAGRAMA UML**

****

**CLASE ARCHIVO TEXTO**

import java.io.\*;

public class ArchivoTexto {

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

try {

PrintWriter salida = new PrintWriter(new BufferedWriter(new FileWriter("prueba1.txt")));

salida.println("Ejemplo de escritura a un archivo de texto");

salida.println("segunda linea");

salida.close();

BufferedReader entrada = new BufferedReader(new FileReader("prueba1.txt"));

String s1, s2 = new String();

while ((s1 = entrada.readLine()) != null) {

s2 += s1 + "\n"; }

System.out.println("Archiv :" + "\n" + s2);

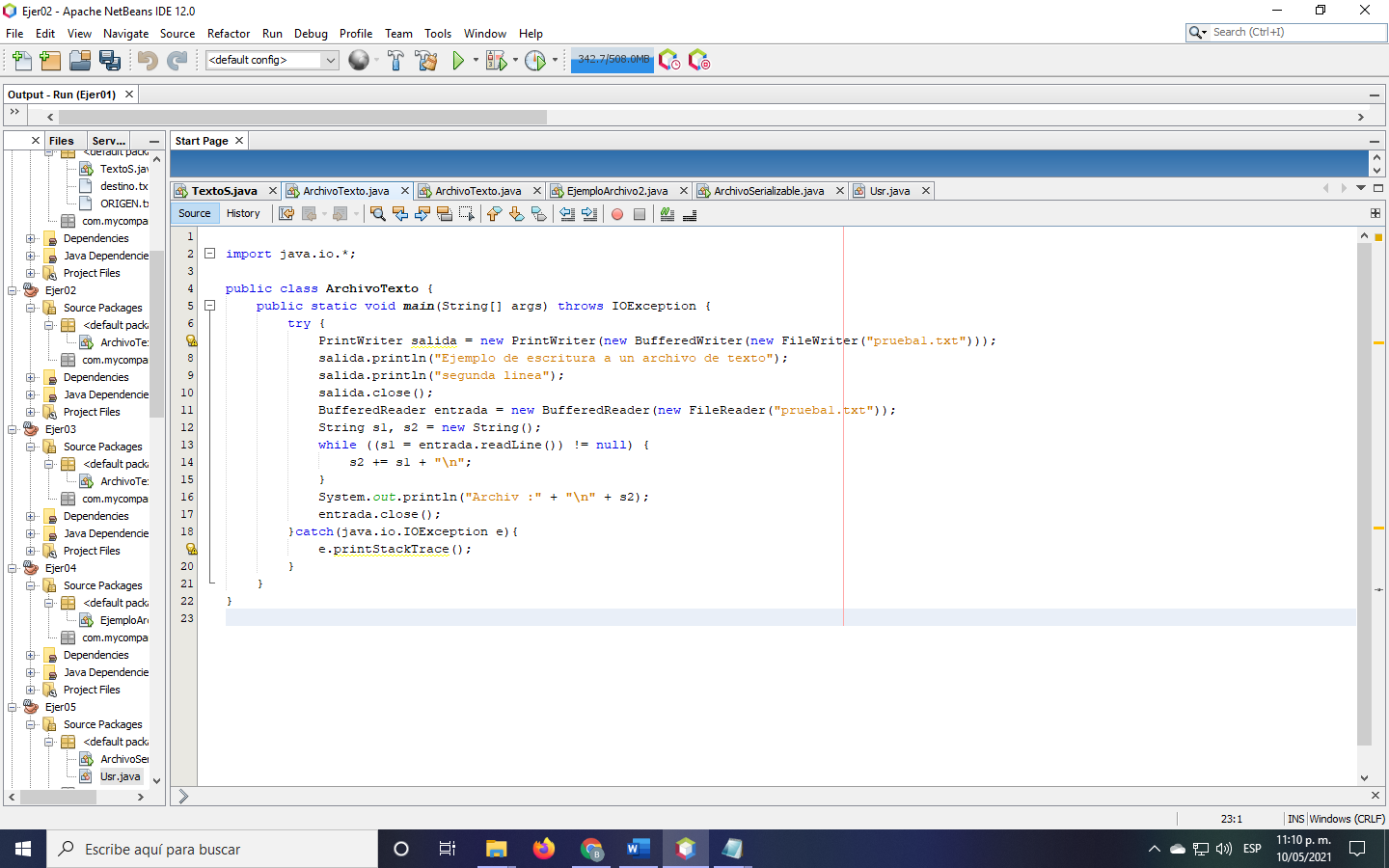
entrada.close();

}catch(java.io.IOException e){

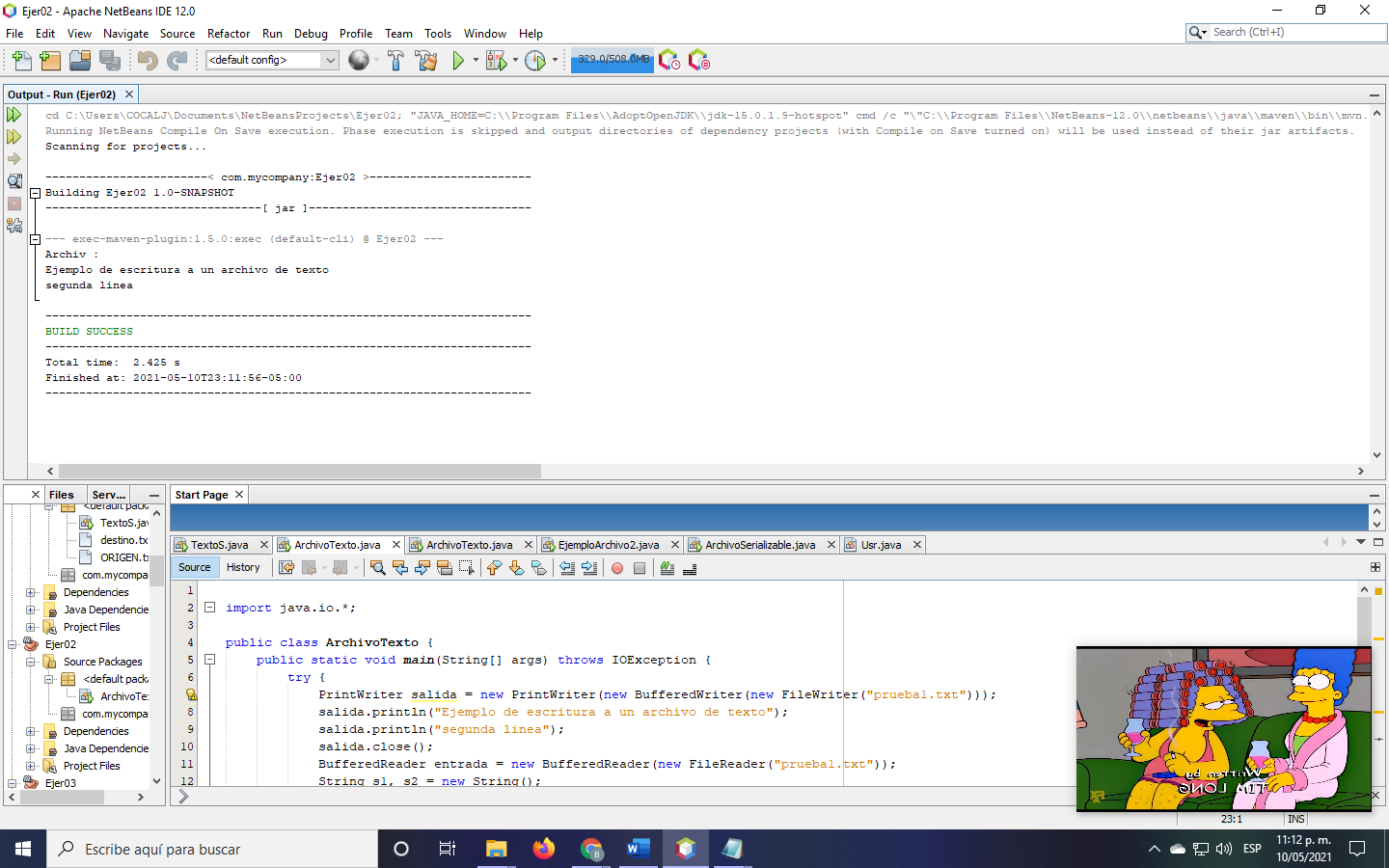
e.printStackTrace(); }

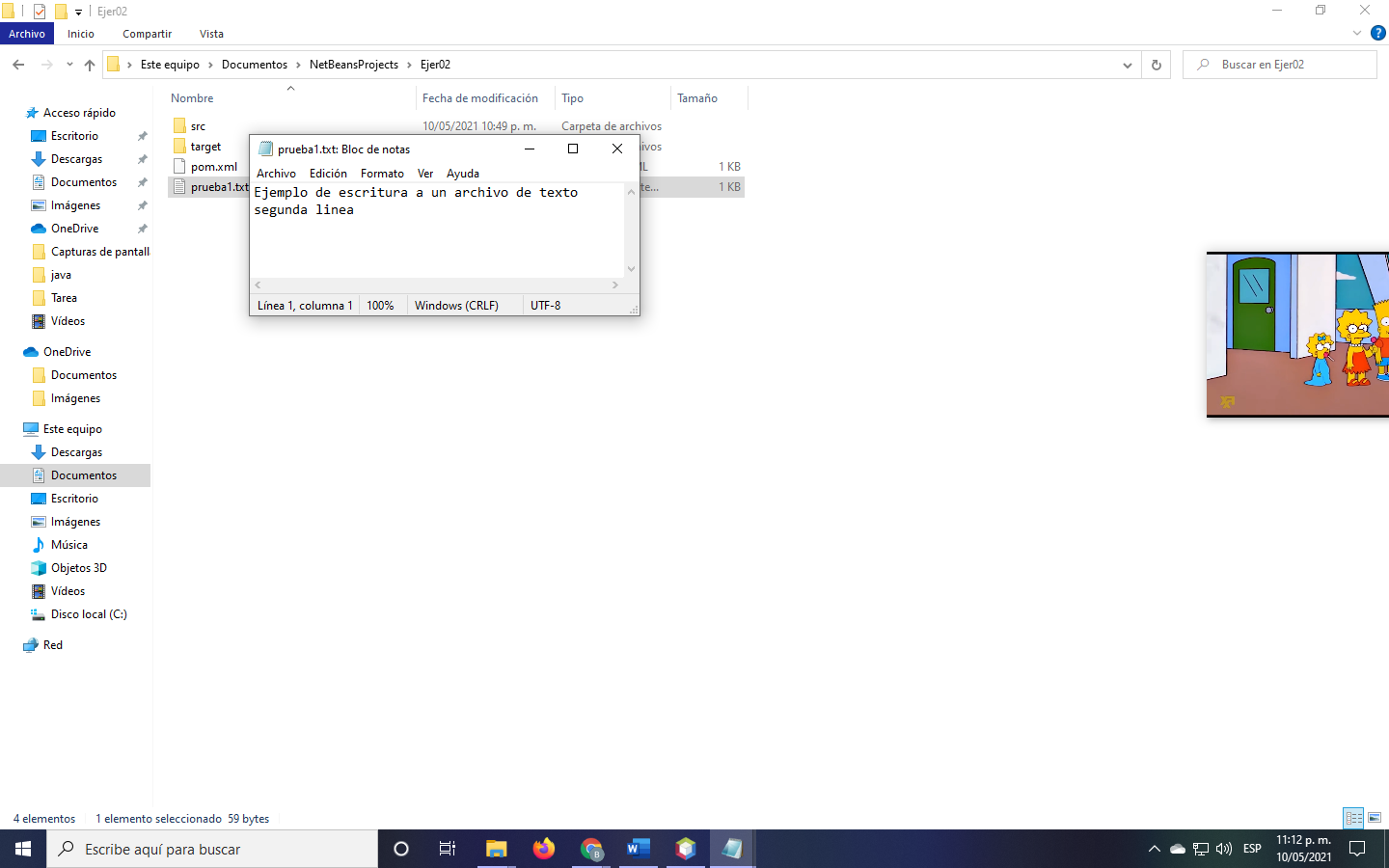
}

}



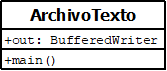
**CORRIDA PROGRAMA**





**PROGRAMA 3**

**DIAGRAMA UML**

****

**CLASE ARCHIVO DE TEXTO**

import java.io.\*;

public class ArchivoTexto {

public static void main(String[] args) {

try{

BufferedWriter out = new BufferedWriter(new FileWriter("datos.txt"));

out.write("esta ");

out.write("es ");

out.write("la primer linea ");

out.newLine();

out.write("Y esta ");

out.write("es la segunda linea ");

out.close();

}catch(java.io.IOException e){

e.printStackTrace();

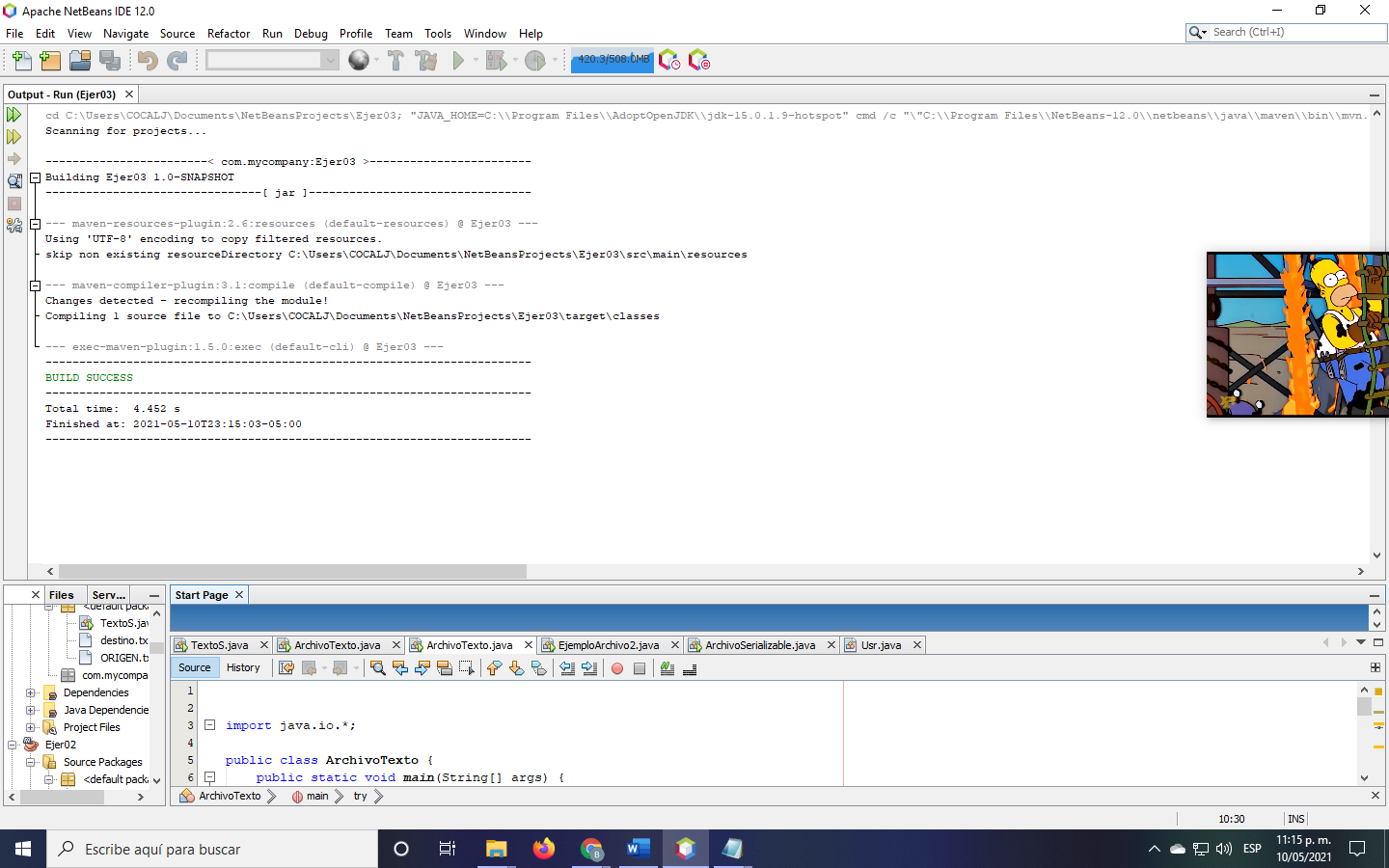
}

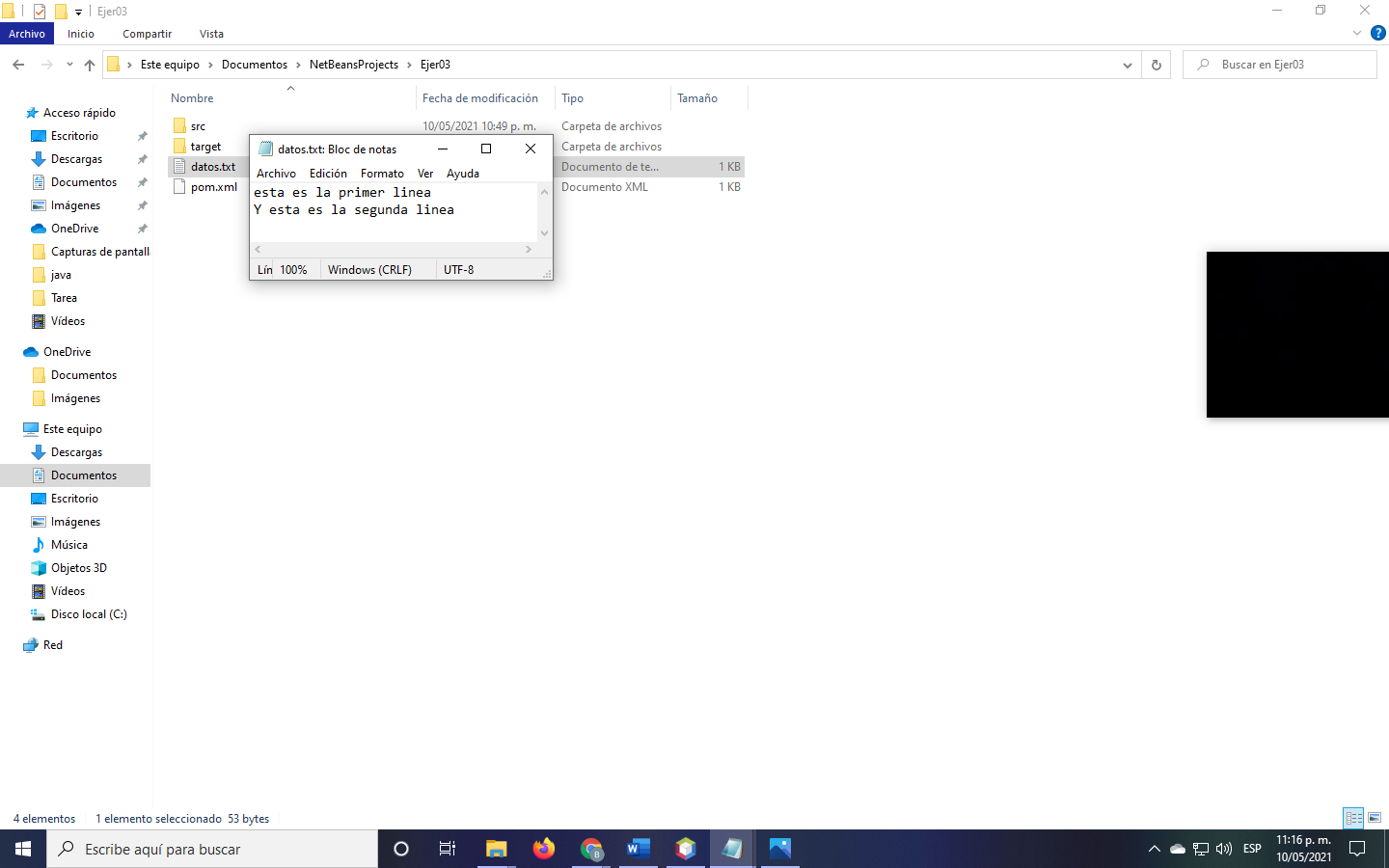
}

}



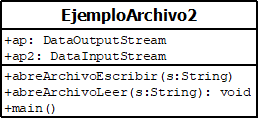
**CORRIDA CODIGO**





**PROGRAMA 4**

**DIAGRAMA UML**

****

**CLASE EJEMPLO ARCHIVO2**

import java.io.\*;

public class EjemploArchivo2 {

static DataOutputStream ap;

static DataInputStream ap2;

public static void abreArchivoEscribir(String s) {

try {

ap = new DataOutputStream(new FileOutputStream(s));

} catch (IOException e) {

System.err.println("No se abrio el archivo" + e.toString());

System.exit(1);

}

}

public static void abreArchivoLeer(String s) {

try {

ap2 = new DataInputStream(new FileInputStream(s));

} catch (IOException e) {

System.err.println("Error .." + e.toString());

System.exit(1);

}

}

public static void main(String s[]) {

abreArchivoEscribir("otroarchivo.dat");

try {

ap.writeInt(5);

ap.writeInt(7);

ap.writeUTF("hola");

ap.writeDouble(4.56);

ap.flush();

ap.close();

} catch (IOException e) {

}

abreArchivoLeer("otroarchivo.dat");

try {

System.out.println(ap2.readInt());

System.out.println(ap2.readInt());

System.out.println(ap2.readUTF());

System.out.println(ap2.readDouble());

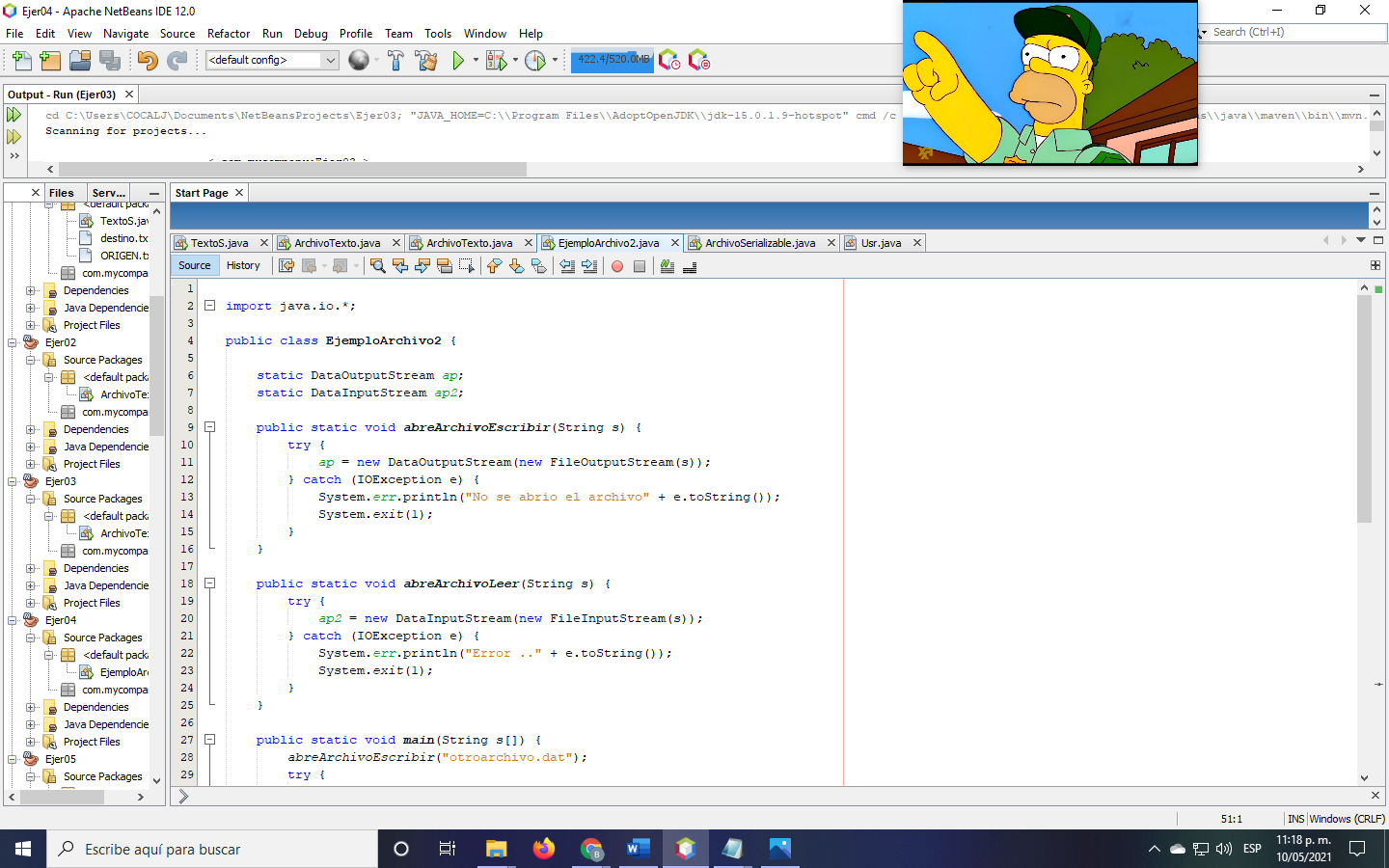
ap2.close();

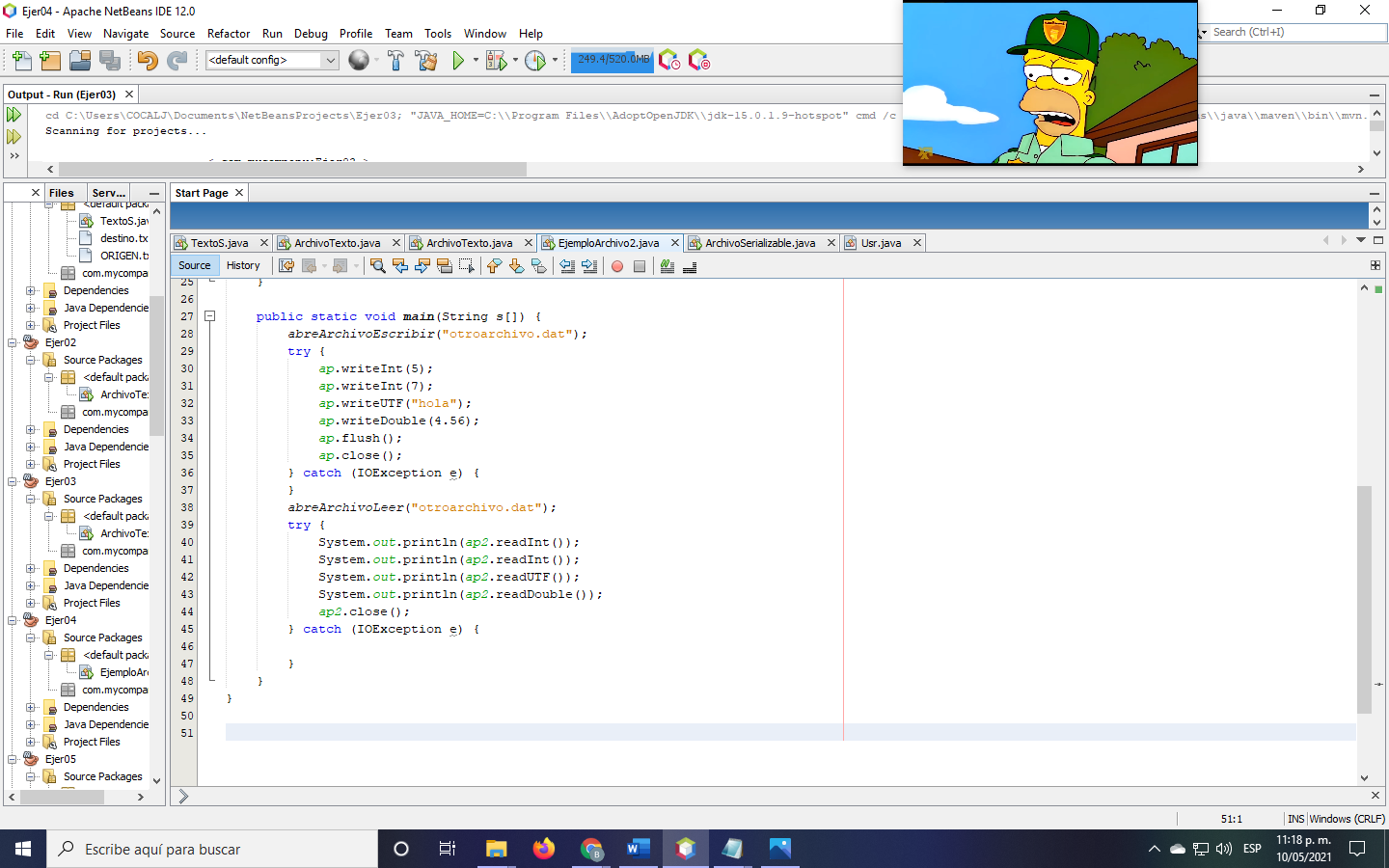
} catch (IOException e) {

}

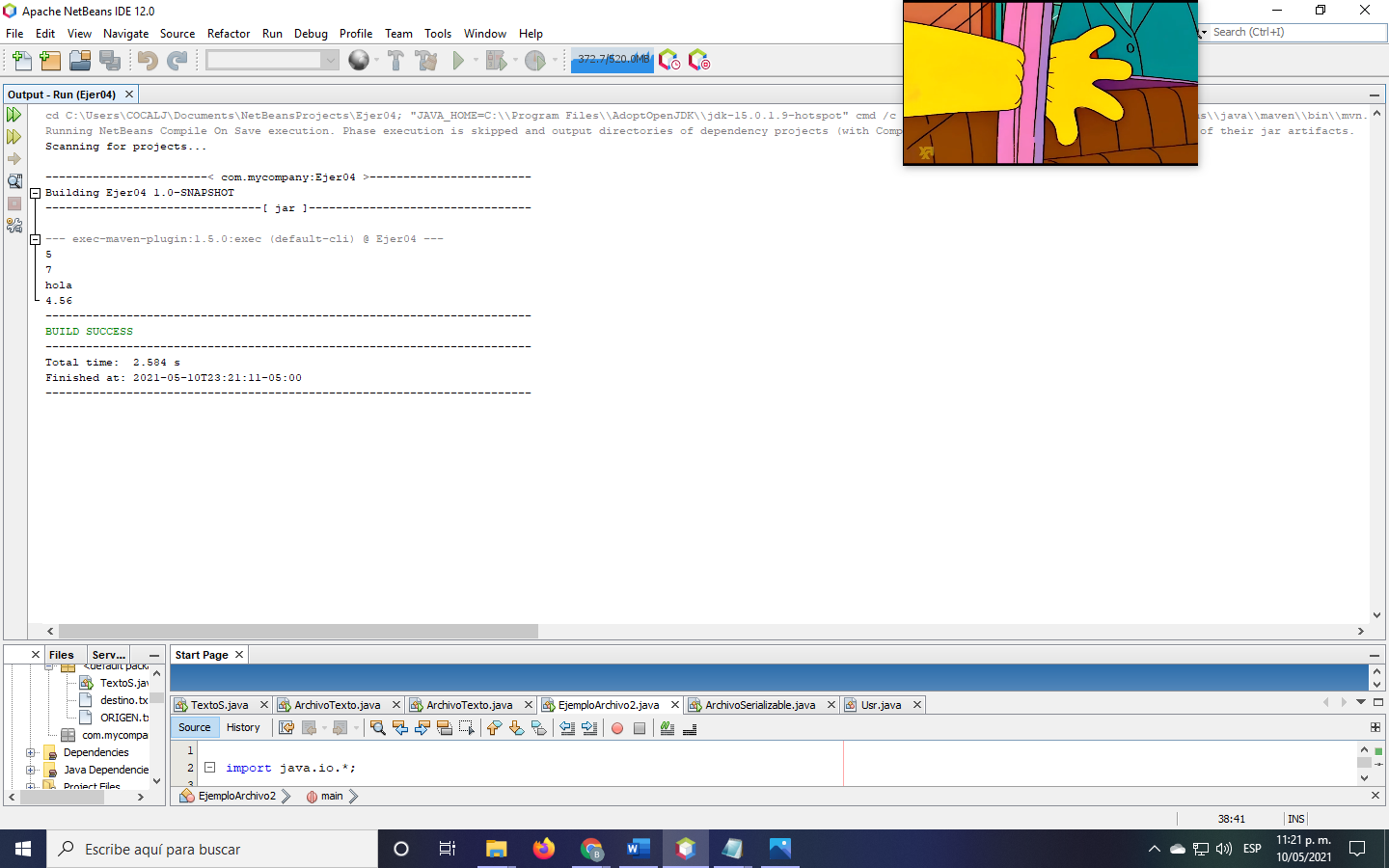
}

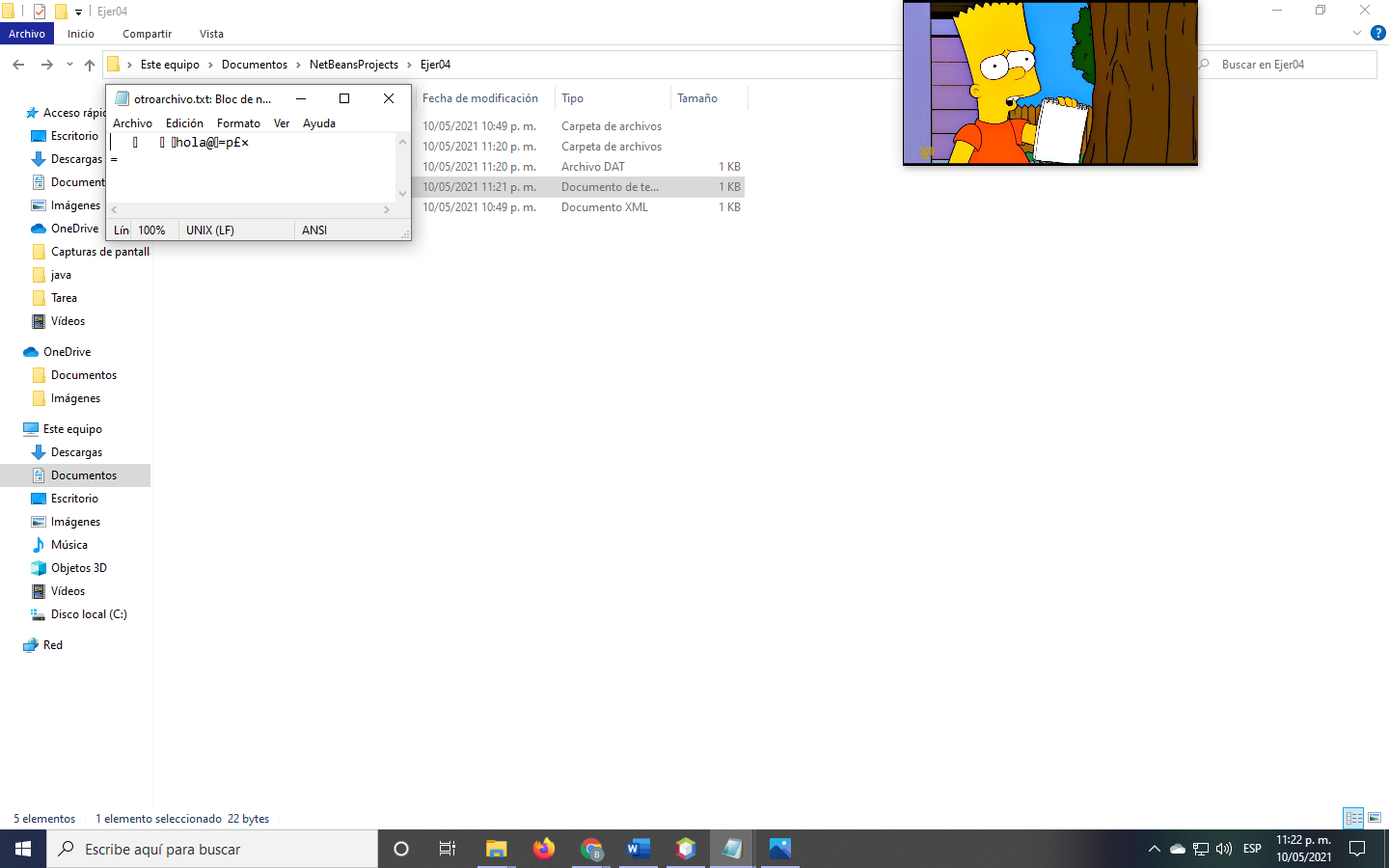
}





**CORRIDA PROGRAMA**





**PROGRAMA 5**

**DIAGRAMA UML**

****

**CLASE ARCHIVOSEREALIZABLE**

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

public class ArchivoSerializable {

public static void main(String[] args) {

String fichero = "objectos.bin";

ObjectOutputStream salidaO = null;

ObjectInputStream entradaO = null;

try{

salidaO = new ObjectOutputStream(new FileOutputStream(fichero));

salidaO.writeObject(new Usr("uno", 1));

salidaO.writeObject(new Usr("dos", 2));

salidaO.writeObject(new Usr("tres", 3));

}catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e){

e.printStackTrace();

}

if (salidaO != null){

try {

salidaO.close();

}catch(IOException e){

}

try{

entradaO = new ObjectInputStream(new FileInputStream(fichero));

System.out.println(entradaO.readObject());

System.out.println(entradaO.readObject());

System.out.println(entradaO.readObject());

}catch (FileNotFoundException e){

e.printStackTrace();

}catch (IOException e) {

e.printStackTrace();

}catch (ClassNotFoundException e) {

e.printStackTrace();

}

if(entradaO != null){

try{

entradaO.close();

}catch(IOException e){

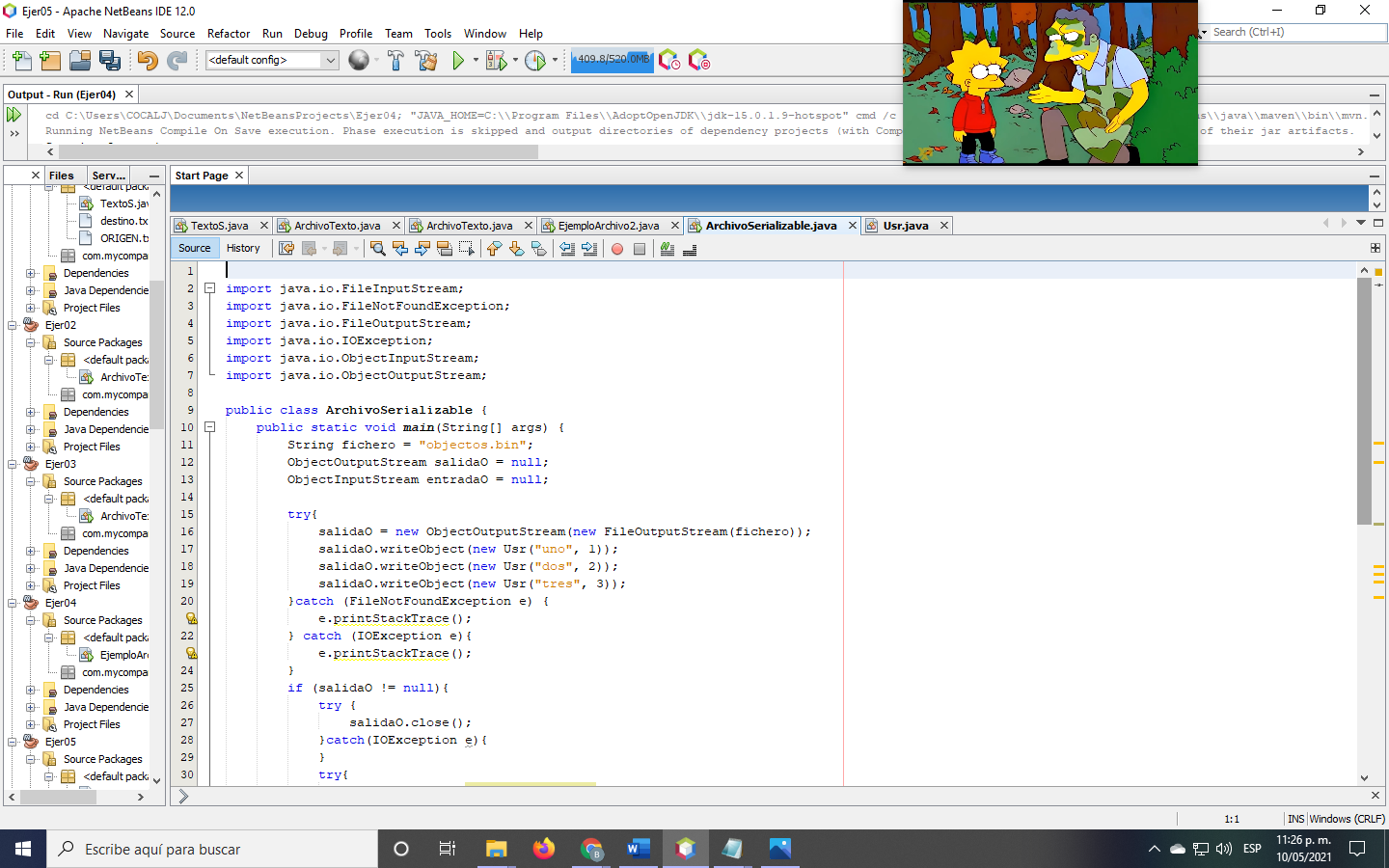
}

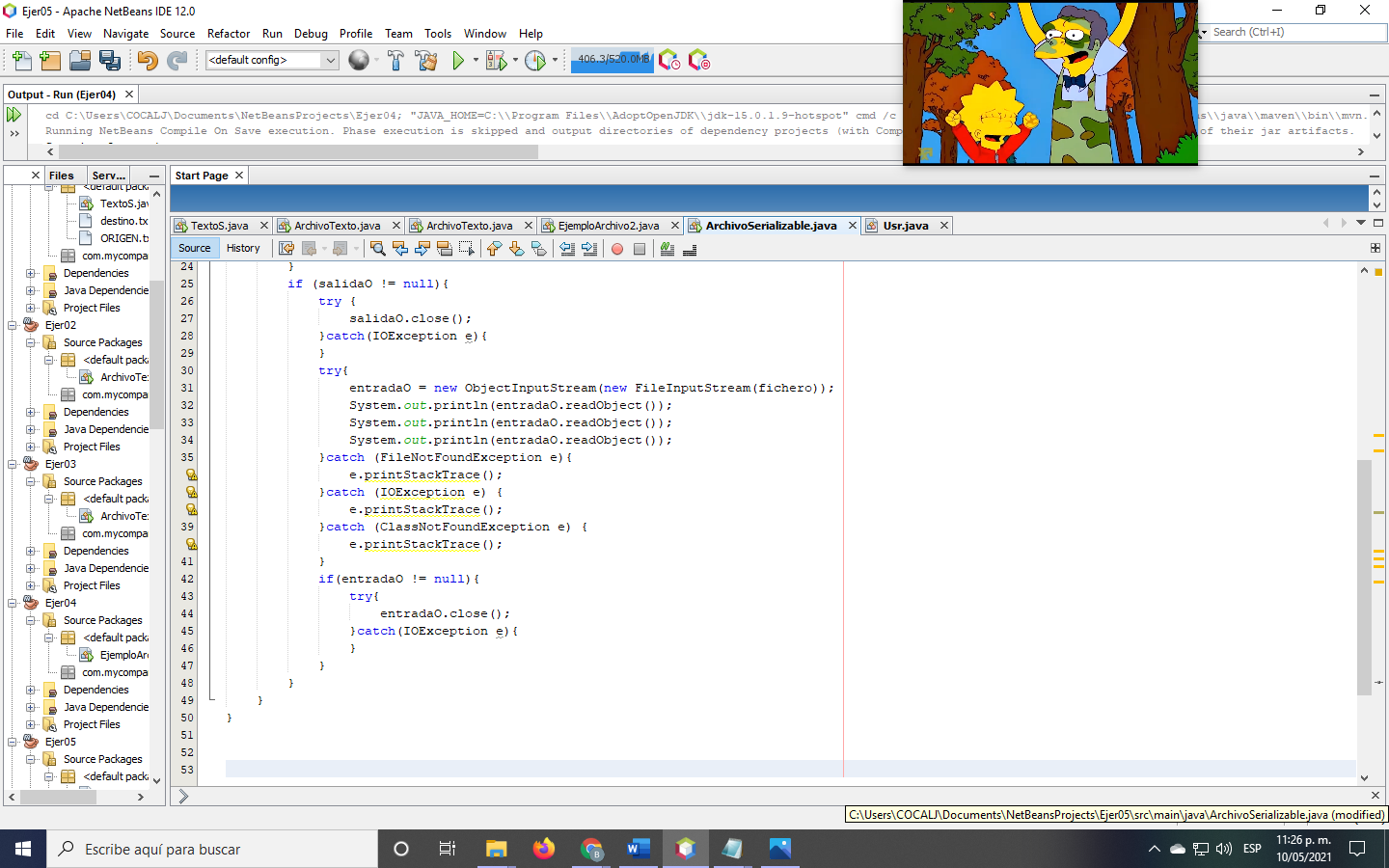
}

}

}

}





**CLASE URS**

import java.io.\*;

public class Usr implements Serializable {

private String nombre;

private long movil;

private int[] c;

public Usr(String nombre, long movil) {

this.nombre = nombre;

this.movil = movil;

c = new int[2];

c[0] = 50;

c[1] = 70;

}

public String getNombre() {

return nombre;

}

public long getMovil() {

return movil;

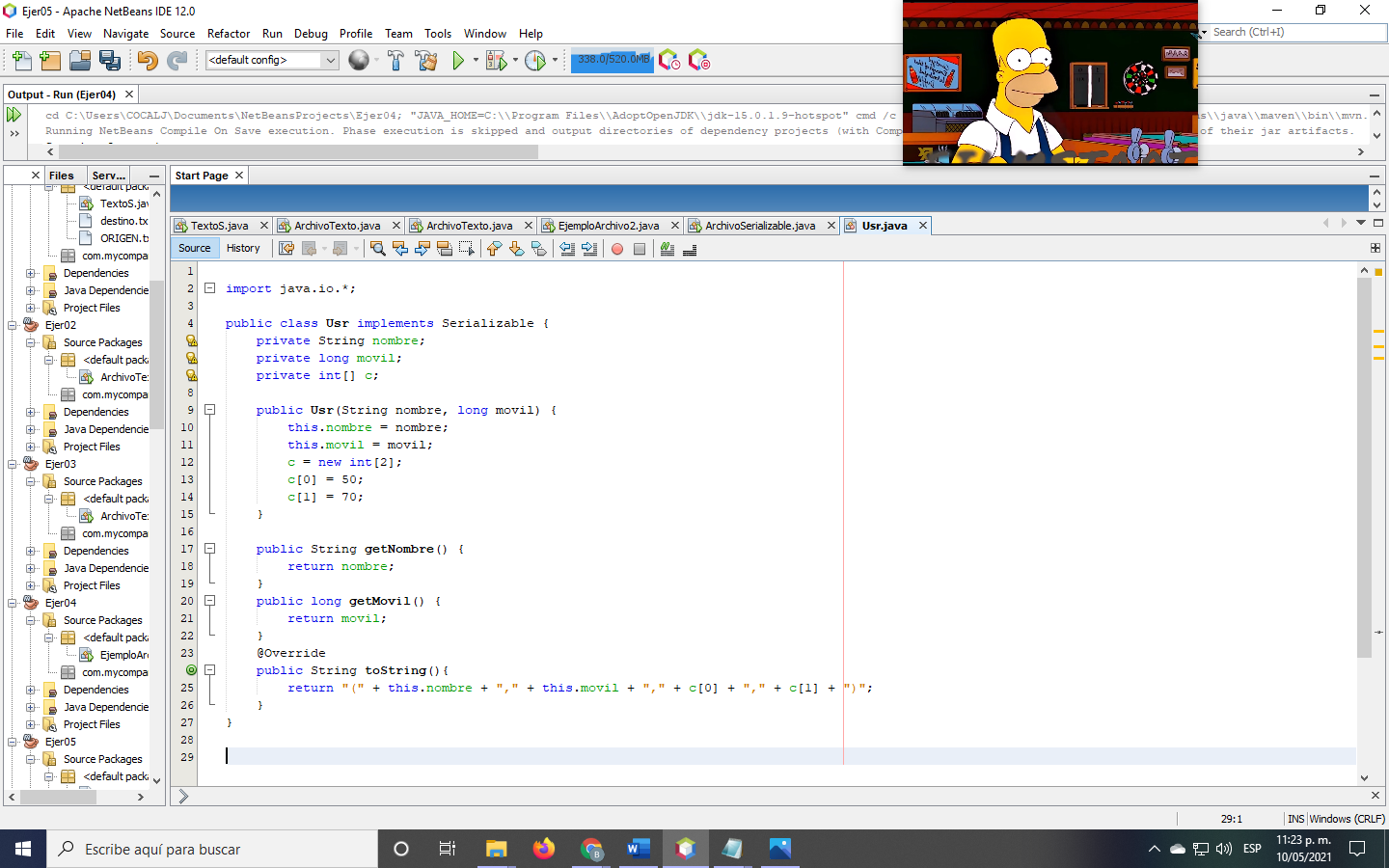
} @Override

public String toString(){

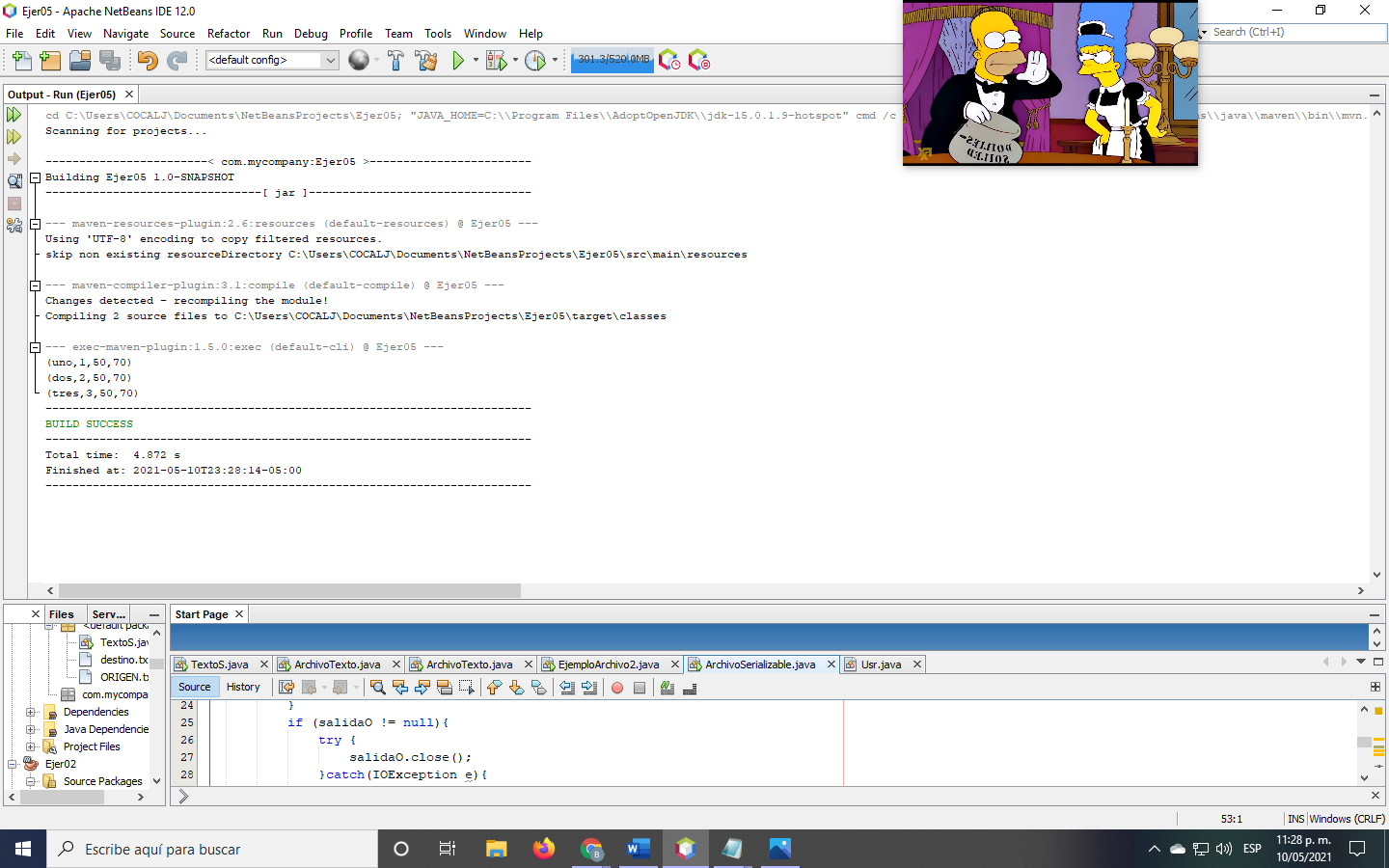
return "(" + this.nombre + "," + this.movil + "," + c[0] + "," + c[1] + ")";

}

}



**CORRIDA PROGRAMA**

****

**CONCLUSIÓN**

Las soluciones recursivas suelen ser más lentas que las iterativas por el tiempo empleado en la gestión de las sucesivas llamadas a los métodos. Además consumen más memoria ya que se deben guardar los contextos de ejecución de cada método que se llama.

A pesar de estos inconvenientes, en ciertos problemas, la recursividad conduce a soluciones que son mucho más fáciles de leer y comprender que su correspondiente solución iterativa. En estos casos una mayor claridad del algoritmo puede compensar el coste en tiempo y en ocupación de memoria.  
De todas maneras, numerosos problemas son difíciles de resolver con soluciones iterativas, y sólo la solución recursiva conduce a la resolución del problema (por ejemplo, Torres de Hanoi o recorrido de Árboles).

**REFERENCIAS BIBLIOGRÁFICAS**

* Fagg, A.H.(2017). GUI Programming: Components. Consultado el día 12 de agosto de 2020 en: <https://www.cs.ou.edu/~fagg/classes/cs2334/lecture/GUI-Components.pdf>
* Horstmann, C. S., & Cornell, G. (2013). Core Java Volumen I Fundamentals. New Jersey: Prentice Hall.
* Joyanes Aguilar, L., & Zahonero Martínez, I. (2011). Programación en Java 6. Algoritmos y programación orientada a objetos. Cd. de México: Mc Graw Hill.
* Poo, D., Kiong, D., & Ashok, S. (2008). Object-Oriented Programming and Java. Singapore: Springer.
* Sznajdleder, P. A. (2016). Java a fondo. Alfaomega.
* Liang, Y. D. ( 2014 ). Introduction to Java Programming, Comprehensive Version. Pearson
* Malik D. S. (2012 ). Java Programming: From Problem Analysis to Program Design, Fifth Edition. Cengage Learning
* GeeksforGeeks. (Última edición: 20 de Julio de 2020) , GeeksforGeeks. https://www.geeksforgeeks.org/types-of-recursions .Consultado de 30 de julio del 2020

**PROGRAMACION II**

**RUBRICA HOJA DE EVALUACIÓN DE PRACTICAS**

**MATRICULA: 201963582 FECHA: 9-MAYO-2021**

**NOMBRE: Irvyn Xicale Cabrera NO. PRACTICA: 16 INDIVIDUAL ( X) COLABORATIVA( )**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Deficiente** | **Suficiente** | **Bueno** | **Excelente** | **Calificación Obtenida** | |
| **CRITERIOS** | **1-5.9** | **6‐7.9** | **8‐9** | **9.1‐10** |
| **%** | **Puntos** |
| **CONOCIMIENTO TEORICO**  **20%** | Conocimiento deficiente de los fundamentos teóricos de POO y no puede aplicarlos en el diseño de clases. | Conocimiento confuso de los fundamentos teóricos de POO y el diseño de las clases y relaciones es incompleto. | Conocimiento claro de los fundamentos teóricos POO, pero requiere mejorar el modelado de las clases, sus métodos  y sus relaciones entre clases. | Dominio del Conocimiento de los fundamentos teóricos POO y puede aplicarlos de forma completa en el modelado de las clases, métodos y todas las relaciones entre clase. |  |  |
| **EJECUCCIÓN DE LA PRACTICA**  **30%** | No puede realizar la práctica ya que desconoce el entorno de trabajo y desarrollo de la práctica en lenguaje UML y Java. | Realiza la práctica de forma incompleta ya que desconoce el entorno de trabajo del lenguaje UML y Java | Realiza la práctica, pero requiere mejorar en el manejo del entorno de trabajo del lenguaje de programación (sintaxis y semántica) | Realiza la práctica de forma correcta y completa, demuestra dominio del entorno de trabajo del lenguaje de programación (sintaxis y semántica). |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SOLUCIÓN DE LA PRACTICA**  **40%** | No puede generar las soluciones o programas a los problemas planteados ya que no posee el dominio teórico y práctico del modelado y el lenguaje de programación. | Propone soluciones confusas o programas incompletos a los problemas planteados, ya que carece del dominio del modelado y lenguaje de programación | Genera soluciones con poca profundidad y los programas no están orientados de acuerdo a los problemas solicitados, por lo cual no tiene un dominio profundo de la temática y del Lenguaje. | Genera soluciones con profundidad y los programas son correctos de acuerdo a los problemas planteados, por lo cual demuestra un dominio de la temática y del Lenguaje de Programación. |  |  |
| **ACTITUD DE APRENDER Y COLABORAR EN EQUIPO DE TRABAJO**  **10%** | No posee una actitud proactiva para un aprendizaje autónomo y no le gusta participar y trabajar en equipo. | Posee una actitud propositiva para un aprendizaje autónomo, participa, pero no le gusta trabajar en equipo. | Posee una actitud propositiva logrando un aprendizaje autónomo, colaborativo, le gusta trabajar en equipo, pero requiere mejorar su participación y portaciones de forma profunda. | Posee una actitud proactiva logrando un aprendizaje autónomo, participa con propuestas concretas y profundas, le gusta trabajar en equipo y asume su responsabilidad dentro para lograr éxito del equipo |  |  |
| **Total** | | | | |  |  |  |

