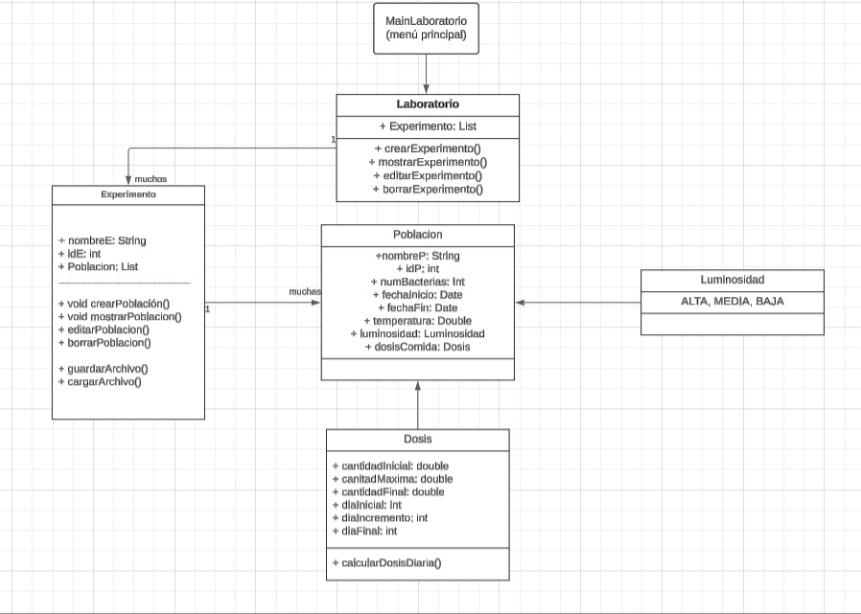
Memoria Practica II

Luis Felipe Riera Aldanondo

**Análisis y Descripción de la aplicación**

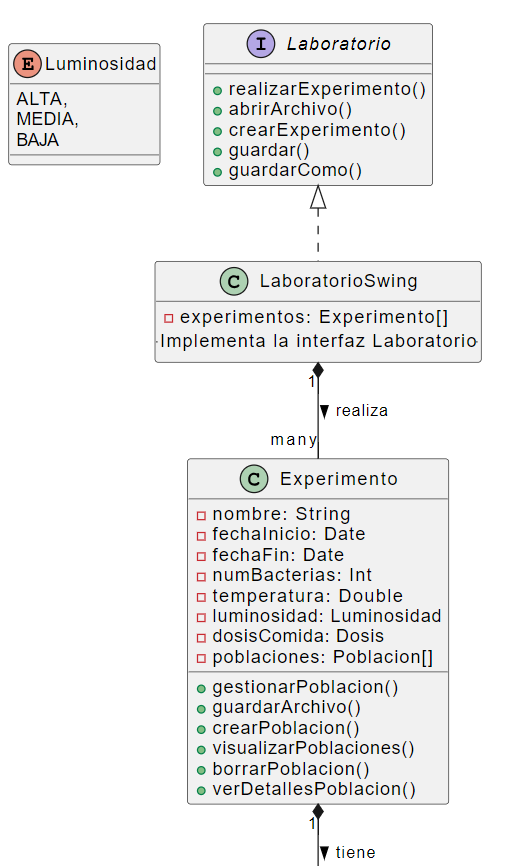
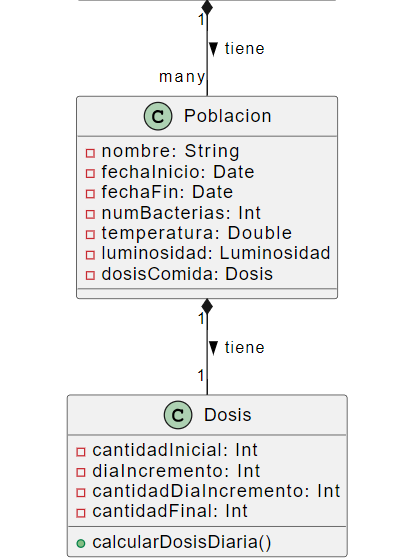
La aplicación es una herramienta que permite a un conjunto de biólogos crear laboratorios, experimentos y poblaciones para recoger la información y almacenarla

UML y estructura de la Práctica

Para plantear la estructura de la practica se ha utilizado el siguiente UML, con variaciones:

Al comienzo de la práctica, este es el diseño que se propuso seguir. Sin embargo, se decidió realizar algunos cambios. El primer cambio es que la clase “MainLaboratorio” cuenta con una lista de laboratorios, ya que distintos laboratorios realizan distintos experimentos; ej: un laboratorio de química se centrará en la sustancia que pueden obtener de cada cultivo de bacterias y un laboratorio de biología se centrará en como es el desarrollo de distintas clases de bacterias en diferentes entornos.

Otro cambio que se ha realizado es que se ha decidido no implementar el método “mostrarExperimento”, ya que no es una funcionalidad que fuese a ser utilizada (el método que se pide es mostrar población). En la clase “Dosis” también he decidido no implementar los atributos “diaInicial” y “diaFinal”, ya que para esta parte de la practica se utilizó otro UML:



Estructuración y organización de las clases

Se han implementado un total de 8 clases, además de tres clases para los tests, y están organizadas en 4 paquetes. El primer paquete, nombrado “Main” contiene la clase “MainLaboratorio” y es la que contiene el menú del programa que hay que ejecutar para usarlo. Su responsabilidad es actuar como punto de entrada y control principal del programa, además de ser la interfaz con el usuario y sigue el siguiente esquema de funciones:

* Presentar un menú de opciones al usuario.
* Leer la opción seleccionada por el usuario utilizando la clase Scanner
* Ejecutar el código correspondiente según la opción seleccionada, invocando métodos de otras clases.
* Ejecutar el código correspondiente según la opción seleccionada, invocando métodos de otras clases.
* Mantener un bucle para permitir al usuario seleccionar múltiples opciones hasta que decida salir.

El segundo paquete se llama “GestionDeMemoria”. GestionDeMemoria esta encargado de la lectura y escritura de los datos del experimento con sistema de archivos. Esto se consigue con los métodos LeerDeMemoria.leerExperimentoDesdeArchivo y GuardarEnMemoria.guardar que se utilizan en el código. Las responsabilidades del paquere se resumen como:

* Lectura de los datos del experimento desde un archivo en el sistema de archivos.
* Guardado de los datos del experimento en un archivo en el sistema de archivos.

Se ha decidido utilizar métodos estáticos para la lectura y escritura de los datos del experimento por dos razones:

* Simplicidad: Los métodos estáticos son más fáciles de usar en algunos casos porque no necesitan de una instancia de la clase para ser llamados. Esto simplifica el código porque no se necesita mantener ningún estado entre las llamadas a estos métodos.
* Estado compartido: Como los datos que se estaban leyendo (y escribiendo) eran compartidos por todas las instancias de la clase (datos globales), entonces tenía sentido que los métodos que los manipulan fuesen estáticos.

En cuanto a comprobaciones de integridad y excepciones, hay múltiples ocasiones en las que se asume que métodos devolverán datos validos, como por ejemplo: se asume que la función “experimento.getPathArchivo()” devolverá un nombre de archivo válido. Si esta función devuelve null, no se realizará ninguna comprobación adicional para este caso específico. Esto es porque no se ha sido muy exhaustivo con el uso y implementación de excepciones. Sin embargo esto no significa que haya ausencia de estas, ya que se utilizan excepciones como IOException, ParseException y FileNotFoundException para realizar algunas comprobaciones.

El tercer paquete se llama “Procesamiento” y es el paquete que se encarga, como su nombre indica de el procesamiento de datos, y tiene la responsabilidad de proporcionar funcionalidades y operaciones relacionadas con el procesamiento y el análisis de datos de las poblaciones, los experimentos y las dosis. Cuenta con 5 clases, que son “Laboratorio”, “Experimento”, “Población”, “Dosis” y “Luminosidad”

* Clase Laboratorio: representa un laboratorio que realiza experimentos.
* Clase Experimento: representa un experimento que se realiza en un laboratorio y contiene poblaciones de bacterias.
* Clase Población: representa una población de bacterias que se utiliza en un experimento.
* Clase Dosis: representa una dosis de comida que se proporciona a una población de bacterias de un experimento.

Fallos conocidos y Funcionalidades no aplicadas

Los test han mostrado algunos errores en el código que no han sido corregidos. El primero se encuentra en la clase “GuardarEnMemoria”. El test “shouldNotCreateExperimentoFromInvalidFileFormat()” muestra que el método “guardar” permite la creación de un experimento desde un archivo de texto no valido, sin lanzar la excepción IllegalArgumentException que se especifica en el try-catch del método. (error: Expected java.lang.IllegalArgumentException to be thrown, but nothing was thrown.)

Otro test de la clase “TestsExperimento”, mas específicamente el test “shouldNotAddNullPoblacion” ha mostrado que el código actual permite la adición de una poblacion nula no resulta en un error o una excepción, sino que se permite. (error: org.opentest4j.AssertionFailedError: Expected :false Actual :true)

Codigo Fuente Ordenado por Paquetes

1. ***Paquete GestionDeMemoria***
   1. *GuardarEnMemoria*

*/\*\*  
 \* The GuardarEnMemoria class provides methods to save an Experimento object to a file.  
 \*/*public class GuardarEnMemoria {  
  
 */\*\*  
 \* Saves an Experimento object to a file with a new name.  
 \* The details of the Experimento and its Poblacion objects are written to the file in a specific format.  
 \* Each detail is on a new line and is in the format "Detail name: Detail value".  
 \* The details of the Experimento are at the top of the file, followed by the details of each Poblacion.  
 \* Each Poblacion is separated by a blank line.  
 \* @param experimento The Experimento object to save  
 \*/* public static void guardarExperimentoComo(Experimento experimento) {  
 String nombreArchivo = experimento.getPathArchivo();  
 try {  
 FileWriter writer = new FileWriter(nombreArchivo);  
  
 *// Write the details of the Experimento* writer.write("Experimento ID: " + experimento.getIdExperimento() + "\n");  
 writer.write("Experimento Name: " + experimento.getNombreExp() + "\n");  
  
 *// Write the details of each Poblacion* for (Poblacion poblacion : experimento.getPoblaciones()) {  
 writer.write("\n");  
 writer.write("Poblacion ID: " + poblacion.getIdPoblacion() + "\n");  
 writer.write("Poblacion Name: " + poblacion.getNombre() + "\n");  
 writer.write("Start Date: " + poblacion.getFechaInicio() + "\n");  
 writer.write("End Date: " + poblacion.getFechaFin() + "\n");  
 writer.write("Number of Bacteria: " + poblacion.getNumBacterias() + "\n");  
 writer.write("Temperature: " + poblacion.getTemperatura() + "\n");  
 writer.write("Luminosity: " + poblacion.getLuminosidad() + "\n");  
 writer.write("Food Dose: " + poblacion.getDosisComida() + "\n");  
 }  
 writer.close();  
 } catch (IOException e) {  
 System.*out*.println("An error occurred while writing to the file.");  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Saves an Experimento object to a file.  
 \* The details of the Experimento and its Poblacion objects are written to the file in a specific format.  
 \* Each detail is on a new line and is in the format "Detail name: Detail value".  
 \* The details of the Experimento are at the top of the file, followed by the details of each Poblacion.  
 \* Each Poblacion is separated by a blank line.  
 \* @param experimento The Experimento object to save  
 \* @return true if the Experimento was saved successfully; false otherwise  
 \*/* public static boolean guardar(Experimento experimento) {  
 String nombreArchivo = experimento.getPathArchivo();  
 try {  
 FileWriter writer = new FileWriter(nombreArchivo);  
  
 *// Write the details of the Experimento* writer.write("Experimento ID: " + experimento.getIdExperimento() + "\n");  
 writer.write("Experimento Name: " + experimento.getNombreExp() + "\n");  
  
 *// Write the details of each Poblacion* for (Poblacion poblacion : experimento.getPoblaciones()) {  
 writer.write("\n");  
 writer.write("Poblacion ID: " + poblacion.getIdPoblacion() + "\n");  
 writer.write("Poblacion Name: " + poblacion.getNombre() + "\n");  
 writer.write("Start Date: " + poblacion.getFechaInicio() + "\n");  
 writer.write("End Date: " + poblacion.getFechaFin() + "\n");  
 writer.write("Number of Bacteria: " + poblacion.getNumBacterias() + "\n");  
 writer.write("Temperature: " + poblacion.getTemperatura() + "\n");  
 writer.write("Luminosity: " + poblacion.getLuminosidad() + "\n");  
 writer.write("Food Dose: " + poblacion.getDosisComida() + "\n");  
 }  
 writer.close();  
 return true; *// Return true if the file was saved correctly* } catch (IOException e) {  
 System.*out*.println("An error occurred while writing to the file.");  
 e.printStackTrace();  
 return false; *// Return false if an exception was thrown* }  
 }  
}

* 1. *LeerDeMemoria*

*/\*\*  
 \* The LeerDeMemoria class provides a method to read an Experimento object from a file.  
 \*/*public class LeerDeMemoria {  
  
 */\*\*  
 \* Reads an Experimento object from a file.  
 \* The file should contain the details of the Experimento and its Poblacion objects in a specific format.  
 \* Each detail should be on a new line and should be in the format "Detail name: Detail value".  
 \* The details of the Experimento should be at the top of the file, followed by the details of each Poblacion.  
 \* Each Poblacion should be separated by a blank line.  
 \* @param pathArchivo The path of the file to read the Experimento from  
 \* @return The Experimento object read from the file, or null if an error occurred  
 \* @throws IllegalArgumentException If the file format is incorrect  
 \*/* public static Experimento leerExperimentoDesdeArchivo(String pathArchivo) {  
 Experimento experimento = null;  
 try {  
 File file = new File(pathArchivo);  
 Scanner scanner = new Scanner(file);  
  
 *// Read the details of the Experimento* int experimentoId = Integer.*parseInt*(scanner.nextLine().split(": ")[1]);  
 String experimentoName = scanner.nextLine().split(": ")[1];  
  
 *// Create the Experimento object* experimento = new Experimento(experimentoId, experimentoName, pathArchivo, new ArrayList<>());  
  
 *// Read the details of each Poblacion* while (scanner.hasNextLine()) {  
 scanner.nextLine();  
 int poblacionId = Integer.*parseInt*(scanner.nextLine().split(": ")[1]);  
 String poblacionName = scanner.nextLine().split(": ")[1];  
 String startDate = scanner.nextLine().split(": ")[1];  
 String endDate = scanner.nextLine().split(": ")[1];  
 int numBacterias = Integer.*parseInt*(scanner.nextLine().split(": ")[1]);  
 float temperatura = Float.*parseFloat*(scanner.nextLine().split(": ")[1]);  
 Luminosidad luminosidad = Luminosidad.*valueOf*(scanner.nextLine().split(": ")[1].toUpperCase());  
 String foodDose = scanner.nextLine().split(": ")[1];  
  
 *// Parse the foodDose string into four double values* String[] foodDoseParts = foodDose.split(",");  
 double part1 = Double.*parseDouble*(foodDoseParts[0]);  
 int part2 = Integer.*parseInt*(foodDoseParts[1]);  
 double part3 = Double.*parseDouble*(foodDoseParts[2]);  
 double part4 = Double.*parseDouble*(foodDoseParts[3]);  
 Dosis dosis = new Dosis(part1, part2, part3, part4);  
 Poblacion poblacion = new Poblacion(poblacionName, poblacionId, new SimpleDateFormat("yyyy-MM-dd").parse(startDate), new SimpleDateFormat("yyyy-MM-dd").parse(endDate), numBacterias, temperatura, luminosidad, dosis);  
  
 *// Add the Poblacion to the Experimento* experimento.agregarPoblacion(poblacion);  
 }  
  
 scanner.close();  
 } catch (FileNotFoundException e) {  
 System.*out*.println("An error occurred while searching the file.");  
 e.printStackTrace();  
 } catch (ParseException e) {  
 System.*out*.println("An error occurred while parsing the date.");  
 e.printStackTrace();  
 } catch (NumberFormatException e) {  
 throw new IllegalArgumentException("The file format is incorrect.", e);  
 }  
 return experimento;  
 }  
}

1. ***Paquete MainLaboratorio***
   1. *MainLaboratorio*

*/\*\*  
 \* The MainLaboratorio class is the main entry point of the application.  
 \* It contains a list of laboratories and a main method that provides a menu for the user to interact with the application.  
 \*/*public class MainLaboratorio {  
 *// List of laboratories* public static List<Laboratorio> *laboratorios*;  
  
 */\*\*  
 \* The main method of the application.  
 \* It provides a menu for the user to interact with the application.  
 \* The user can choose to open an experiment file, create a new experiment, create a bacteria population, view all bacteria populations,  
 \* delete a bacteria population, view detailed information of a bacteria population, save the current state, save the current state as a new file, or exit the application.  
 \* @param args The command line arguments  
 \*/* public static void main(String[] args) {  
 *// Create a Scanner object for user input* Scanner scanner = new Scanner(System.*in*);  
 *// Variable to store the user's menu choice* int option;  
  
 *// Loop until the user chooses to exit* do {  
 *// Print the menu* System.*out*.println("Menu:");  
 System.*out*.println("1. Abrir un archivo que contenga un experimento");  
 System.*out*.println("2. Crear un nuevo experimento");  
 System.*out*.println("3. Crear una población de bacterias y añadirla al experimento actual");  
 System.*out*.println("4. Visualizar los nombres de todas las poblaciones de bacterias del experimento actual");  
 System.*out*.println("5. Borrar una población de bacterias del experimento actual");  
 System.*out*.println("6. Ver información detallada de una población de bacterias del experimento actual");  
 System.*out*.println("7. Guardar");  
 System.*out*.println("8. Guardar como");  
 System.*out*.println("0. Salir");  
  
 *// Prompt the user to enter a choice* System.*out*.print("Ingrese la opción deseada: ");  
 option = scanner.nextInt();  
 scanner.nextLine();  
  
 *// Perform an action based on the user's choice* switch (option) {  
 case 1:  
 System.*out*.println("Opción 1 seleccionada: Abrir un archivo que contenga un experimento");  
 System.*out*.println("Ingrese la ruta del archivo:");  
 String path = scanner.nextLine();  
 Experimento experimento = LeerDeMemoria.*leerExperimentoDesdeArchivo*(path);  
 if (experimento != null) {  
 System.*out*.println("Experimento cargado con éxito.");  
 System.*out*.println(experimento);  
 } else {  
 System.*out*.println("No se pudo cargar el experimento.");  
 }  
  
 break;  
 case 2:  
 System.*out*.println("Opción 2 seleccionada: Crear un nuevo experimento");  
  
 if (*laboratorios*.isEmpty()) {  
 System.*out*.println("No hay laboratorios creados. Por favor, cree uno.");  
 System.*out*.println("Ingrese el nombre del laboratorio:");  
 String nombreLab = scanner.nextLine();  
 Laboratorio newLaboratorio = new Laboratorio(nombreLab);  
 *laboratorios*.add(newLaboratorio);  
 System.*out*.println("Laboratorio creado con éxito.");  
 }  
 System.*out*.println("Ingrese el nombre del laboratorio en el que desea crear el experimento:");  
 String nombreLab = scanner.nextLine();  
 Laboratorio laboratorio = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab)) {  
 laboratorio = lab;  
 break;  
 }  
 }  
  
 if (laboratorio == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.println("Ingrese el nombre del experimento:");  
 String nombreExp = scanner.nextLine();  
  
 System.*out*.println("Ingrese la ruta del archivo:");  
 String pathArchivo = scanner.nextLine();  
 ArrayList<Poblacion> poblaciones = new ArrayList<>();  
  
 laboratorio.crearExperimento(idExperimento, nombreExp, pathArchivo, poblaciones);  
 System.*out*.println("Experimento creado con éxito en el laboratorio " + nombreLab + ".");  
 break;  
 case 3:  
 System.*out*.println("Opción 3 seleccionada: Crear una población de bacterias y añadirla a un experimento");  
  
 System.*out*.println("Ingrese el nombre del laboratorio en el que desea crear la población:");  
 String nombreLabCase3 = scanner.nextLine();  
 Laboratorio laboratorioCase3 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLabCase3)) {  
 laboratorioCase3 = lab;  
 break;  
 }  
 }  
 if (laboratorioCase3 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 if (laboratorioCase3.getExperimentos().isEmpty()) {  
 System.*out*.println("No hay experimentos creados en este laboratorio. Por favor, cree uno.");  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimentoCase3 = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.println("Ingrese el nombre del experimento:");  
 String nombreExp2 = scanner.nextLine();  
  
 System.*out*.println("Ingrese la ruta del archivo:");  
 String pathArchivo2= scanner.nextLine();  
 ArrayList<Poblacion> poblaciones2 = new ArrayList<>();  
  
 laboratorioCase3.crearExperimento(idExperimentoCase3, nombreExp2, pathArchivo2, poblaciones2);  
 System.*out*.println("Experimento creado con éxito en el laboratorio " + nombreLabCase3 + ".");  
 }  
  
 System.*out*.println("Ingrese el ID del experimento en el que desea crear la población:");  
 int idExperimentoCase3 = scanner.nextInt();  
 scanner.nextLine();  
  
 Experimento experimentoCase3 = null;  
 for (Experimento exp : laboratorioCase3.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimentoCase3) {  
 experimentoCase3 = exp;  
 break;  
 }  
 }  
 if (experimentoCase3 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID de la población:");  
 int idPoblacion = scanner.nextInt();  
 scanner.nextLine();  
  
 System.*out*.println("Ingrese el nombre de la población:");  
 String nombrePoblacion = scanner.nextLine();  
  
 Date fechaInicio = null;  
 Date fechaFin = null;  
 try {  
 System.*out*.println("Ingrese la fecha de inicio (yyyy-MM-dd):");  
 String fechaInicioStr = scanner.nextLine();  
 fechaInicio = new SimpleDateFormat("yyyy-MM-dd").parse(fechaInicioStr);  
  
 System.*out*.println("Ingrese la fecha de fin (yyyy-MM-dd):");  
 String fechaFinStr = scanner.nextLine();  
 fechaFin = new SimpleDateFormat("yyyy-MM-dd").parse(fechaFinStr);  
 } catch (ParseException e) {  
 System.*out*.println("Invalid date format. Please enter the date in the format yyyy-MM-dd.");  
 }  
  
 System.*out*.println("Ingrese el número de bacterias:");  
 int numBacterias = scanner.nextInt();  
  
 System.*out*.println("Ingrese la temperatura:");  
 float temperatura = scanner.nextFloat();  
  
 System.*out*.println("Ingrese la luminosidad (ALTA, MEDIA, BAJA):");  
 Luminosidad luminosidad = Luminosidad.*valueOf*(scanner.next().toUpperCase());  
  
 System.*out*.println("Ingrese la dosis de comida (cantidad inicial, día de incremento, cantidad de incremento, cantidad final):");  
 double cantidadInicial = scanner.nextDouble();  
 int diaDeIncremento = scanner.nextInt();  
 double cantidadIncremento = scanner.nextDouble();  
 double cantidadFinal = scanner.nextDouble();  
 Dosis dosisComida = new Dosis(cantidadInicial, diaDeIncremento, cantidadIncremento, cantidadFinal);  
  
 Poblacion poblacionCase3 = new Poblacion(nombrePoblacion, idPoblacion, fechaInicio, fechaFin, numBacterias, temperatura, luminosidad, dosisComida);  
 experimentoCase3.getPoblaciones().add(poblacionCase3);  
 System.*out*.println("Población creada con éxito en el experimento " + idExperimentoCase3 + ".");  
 break;  
 case 4:  
 System.*out*.println("Opción 4 seleccionada: Visualizar los nombres de todas las poblaciones de bacterias de un experimento ");  
  
 System.*out*.println("Ingrese el nombre del laboratorio con el experimento que desea ver:");  
 String nombreLab2 = scanner.nextLine();  
  
 Laboratorio laboratorio2 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab2)) {  
 laboratorio2 = lab;  
 break;  
 }  
 }  
 if (laboratorio2 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento2 = scanner.nextInt();  
 scanner.nextLine();  
  
 Experimento experimento2 = null;  
 for (Experimento exp : laboratorio2.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimento2) {  
 experimento2 = exp;  
 break;  
 }  
 }  
 if (experimento2 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
  
 for (Poblacion poblacion : experimento2.getPoblaciones()) {  
 System.*out*.println("Poblacion ID: " + poblacion.getIdPoblacion() + ", Nombre: " + poblacion.getNombre());  
 }  
 break;  
 case 5:  
 System.*out*.println("Opción 5 seleccionada: Borrar una población de bacterias del experimento actual");  
  
 System.*out*.println("Ingrese el nombre del laboratorio:");  
 String nombreLab3 = scanner.nextLine();  
  
 *// Find the laboratory with the given name* Laboratorio laboratorio3 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab3)) {  
 laboratorio3 = lab;  
 break;  
 }  
 }  
  
 if (laboratorio3 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento3 = scanner.nextInt();  
 scanner.nextLine(); *// Consume the newline left-over  
  
 // Find the experiment with the given ID* Experimento experimento3= null;  
 for (Experimento exp : laboratorio3.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimento3) {  
 experimento3 = exp;  
 break;  
 }  
 }  
  
 if (experimento3 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID de la población que desea borrar:");  
 int idPoblacion3 = scanner.nextInt();  
 scanner.nextLine();  
  
 Poblacion poblacionToRemove = null;  
 for (Poblacion poblacion : experimento3.getPoblaciones()) {  
 if (poblacion.getIdPoblacion() == idPoblacion3) {  
 poblacionToRemove = poblacion;  
 break;  
 }  
 }  
  
 if (poblacionToRemove == null) {  
 System.*out*.println("No se encontró la población con el ID dado.");  
 } else {  
 experimento3.borrarPoblacion(poblacionToRemove);  
 System.*out*.println("Población borrada con éxito del experimento " + idExperimento3 + ".");  
 }  
 break;  
 case 6:  
 System.*out*.println("Opción 6 seleccionada: Ver información detallada de una población de bacterias de un experimento");  
  
 System.*out*.println("Ingrese el nombre del laboratorio:");  
 String nombreLab6 = scanner.nextLine();  
 Laboratorio laboratorio6 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab6)) {  
 laboratorio6 = lab;  
 break;  
 }  
 }  
 if (laboratorio6 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento6 = scanner.nextInt();  
 scanner.nextLine();  
 Experimento experimento6 = null;  
 for (Experimento exp : laboratorio6.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimento6) {  
 experimento6 = exp;  
 break;  
 }  
 }  
 if (experimento6 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el nombre de la población que desea ver:");  
 String nombrePoblacion6 = scanner.nextLine();  
 String result6 = experimento6.mostrarPoblacion(nombrePoblacion6);  
 System.*out*.println(result6);  
 Poblacion poblacion6 = null;  
 for (Poblacion poblacion : experimento6.getPoblaciones()) {  
 if (poblacion.getNombre().equals(nombrePoblacion6)) {  
 poblacion6 = poblacion;  
 break;  
 }  
 }  
 if (poblacion6 != null) {  
 Dosis dosis = poblacion6.getDosisComida();  
 dosis.calcularDosisDiaria(dosis);  
 } else {  
 System.*out*.println("No se encontró la población con el nombre dado.");  
 }  
  
 System.*out*.println(result6);  
 break;  
 case 7:  
 System.*out*.println("Opción 7 seleccionada: Guardar");  
 System.*out*.println("Ingrese el nombre del laboratorio:");  
 String nombreLab7 = scanner.nextLine();  
 Laboratorio laboratorio7 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab7)) {  
 laboratorio7 = lab;  
 break;  
 }  
 }  
 if (laboratorio7 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento7 = scanner.nextInt();  
 scanner.nextLine();  
 Experimento experimento7 = null;  
 for (Experimento exp : laboratorio7.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimento7) {  
 experimento7 = exp;  
 break;  
 }  
 }  
 if (experimento7 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
  
 boolean result7 = GuardarEnMemoria.*guardar*(experimento7);  
 if (result7) {  
 System.*out*.println("Experimento guardado con éxito.");  
 } else {  
 System.*out*.println("Hubo un error al guardar el experimento.");  
 }  
 break;  
 case 8:  
 System.*out*.println("Opción 8 seleccionada: Guardar como");  
 System.*out*.println("Ingrese el nombre del laboratorio:");  
 String nombreLab8 = scanner.nextLine();  
 Laboratorio laboratorio8 = null;  
 for (Laboratorio lab : *laboratorios*) {  
 if (lab.getLabName().equals(nombreLab8)) {  
 laboratorio8 = lab;  
 break;  
 }  
 }  
 if (laboratorio8 == null) {  
 System.*out*.println("No se encontró el laboratorio con el nombre dado.");  
 break;  
 }  
  
 System.*out*.println("Ingrese el ID del experimento:");  
 int idExperimento8 = scanner.nextInt();  
 scanner.nextLine();  
  
 Experimento experimento8 = null;  
 for (Experimento exp : laboratorio8.getExperimentos()) {  
 if (exp.getIdExperimento() == idExperimento8) {  
 experimento8 = exp;  
 break;  
 }  
 }  
 if (experimento8 == null) {  
 System.*out*.println("No se encontró el experimento con el ID dado.");  
 break;  
 }  
 System.*out*.println("Ingrese la nueva ruta del archivo:");  
 String newPath = scanner.nextLine();  
  
 experimento8.setPathArchivo(newPath);  
 boolean result8 = GuardarEnMemoria.*guardar*(experimento8);  
 if (result8) {  
 System.*out*.println("Experimento guardado con éxito en la nueva ruta.");  
 } else {  
 System.*out*.println("Hubo un error al guardar el experimento en la nueva ruta.");  
 }  
  
 break;  
 case 0:  
 System.*out*.println("Saliendo del programa...");  
 break;  
 default:  
 System.*out*.println("Opción inválida. Intente de nuevo.");  
 break;  
 }  
 } while (option != 0);  
 *// Close the scanner* scanner.close();  
 }  
}

1. *Paquete Procesamiento*
   1. *Laboratorio*

*/\*\*  
 \* The Laboratorio class represents a laboratory in which experiments are conducted.  
 \* It contains a list of Experimento objects, each representing an individual experiment.  
 \*/*public class Laboratorio {  
 *// The name of the laboratory* public String labName;  
 *// The list of experiments conducted in the laboratory* public List<Experimento> experimentos;  
  
 */\*\*  
 \* Constructs a new Laboratorio with the given name.  
 \* @param labName The name of the laboratory  
 \*/* public Laboratorio(String labName) {  
 this.labName = labName;  
 this.experimentos = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Creates a new Experimento and adds it to the list of experiments.  
 \* @param idExperimento The ID of the new experiment  
 \* @param nombreExp The name of the new experiment  
 \* @param pathArchivo The path to the file associated with the new experiment  
 \* @param poblaciones The list of populations involved in the new experiment  
 \*/* public void crearExperimento(int idExperimento, String nombreExp, String pathArchivo, ArrayList<Poblacion> poblaciones) {  
 Experimento newExperimento = new Experimento(idExperimento, nombreExp, pathArchivo, poblaciones);  
 this.experimentos.add(newExperimento);  
 }  
  
 */\*\*  
 \* Edits an existing Experimento based on the user's choice.  
 \* @param choice The index of the experiment to be edited  
 \* @param newIdStr The new ID for the experiment  
 \* @param newName The new name for the experiment  
 \* @return A string indicating the result of the operation  
 \*/* public String editarExperimento(int choice, String newIdStr, String newName) {  
 StringBuilder output = new StringBuilder();  
  
 if (choice > 0 && choice <= experimentos.size()) {  
 Experimento experimento = experimentos.get(choice - 1);  
  
 if (newIdStr != null) {  
 try {  
 int newId = Integer.*parseInt*(newIdStr);  
 experimento.setIdExperimento(newId);  
 } catch (NumberFormatException e) {  
 output.append("Invalid ID format. Please enter a valid integer.\n");  
 }  
 }  
  
 if (newName != null) {  
 experimento.setNombreExp(newName);  
 }  
  
 } else {  
 output.append("Invalid choice. Please enter a number between 1 and " + experimentos.size() + "\n");  
 }  
  
 return output.toString();  
 }  
  
 */\*\*  
 \* Deletes an Experimento from the list of experiments based on its ID.  
 \* @param id The ID of the experiment to be deleted  
 \* @return A boolean indicating whether the operation was successful  
 \*/* public boolean eliminarExperimentoPorId(int id) {  
 for (Experimento experimento : experimentos) {  
 if (experimento.getIdExperimento() == id) {  
 experimentos.remove(experimento);  
 return true;  
 }  
 }  
 return false;  
 }  
  
 *// Getters and setters for the Laboratorio's fields* public String getLabName() {  
 return labName;  
 }  
  
 public void setLabName(String labName) {  
 this.labName = labName;  
 }  
  
 public List<Experimento> getExperimentos() {  
 return experimentos;  
 }  
  
 public void setExperimentos(List<Experimento> experimentos) {  
 this.experimentos = experimentos;  
 }  
}

* 1. *Experimento*

*/\*\*  
 \* The Experimento class represents an experiment in a laboratory.  
 \* It contains information about the experiment such as its id, name, file path, and a list of populations involved in the experiment.  
 \*/*public class Experimento {  
 *// The id of the experiment* public int idExperimento;  
 *// The name of the experiment* public String nombreExp;  
 *// The file path of the experiment* public String pathArchivo;  
 *// The list of populations involved in the experiment* public ArrayList<Poblacion> poblaciones;  
  
 */\*\*  
 \* Constructs a new Experimento with the given parameters.  
 \* @param idExperimento The id of the experiment  
 \* @param nombreExp The name of the experiment  
 \* @param pathArchivo The file path of the experiment  
 \* @param poblaciones The list of populations involved in the experiment  
 \*/* public Experimento(int idExperimento, String nombreExp, String pathArchivo, List<Poblacion> poblaciones) {  
 this.idExperimento = idExperimento;  
 this.nombreExp = nombreExp;  
 this.pathArchivo = pathArchivo;  
 this.poblaciones = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Adds a population to the list of populations involved in the experiment.  
 \* @param poblacion The population to be added  
 \*/* public void agregarPoblacion(Poblacion poblacion) {  
 this.poblaciones.add(poblacion);  
 }  
  
 */\*\*  
 \* Creates a new population and adds it to the list of populations involved in the experiment.  
 \* @param nombre The name of the population  
 \* @param idPoblacion The id of the population  
 \* @param fechaInicio The start date of the population  
 \* @param fechaFin The end date of the population  
 \* @param numBacterias The number of bacteria in the population  
 \* @param temperatura The temperature of the population  
 \* @param luminosidad The luminosity of the population  
 \* @param dosisComida The food dose of the population  
 \*/* public void crearPoblacion(String nombre, int idPoblacion, Date fechaInicio, Date fechaFin, int numBacterias, float temperatura, Luminosidad luminosidad, Dosis dosisComida) {  
 Poblacion poblacion1 = new Poblacion(nombre, idPoblacion, fechaInicio, fechaFin, numBacterias, temperatura, luminosidad, dosisComida);  
 agregarPoblacion(poblacion1);  
 }  
  
 */\*\*  
 \* Removes a population from the list of populations involved in the experiment.  
 \* @param poblacion The population to be removed  
 \* @return true if the population was successfully removed; false otherwise  
 \*/* public boolean borrarPoblacion(Poblacion poblacion) {  
 return this.poblaciones.remove(poblacion);  
 }  
  
 */\*\*  
 \* Returns a string representation of a population with the given name.  
 \* @param nombrePoblacion The name of the population  
 \* @return A string representation of the population if it exists; a message indicating that the population was not found otherwise  
 \*/* public String mostrarPoblacion(String nombrePoblacion) {  
 Poblacion poblacion = null;  
 for (Poblacion p : poblaciones) {  
 if (p.getNombre().equals(nombrePoblacion)) {  
 poblacion = p;  
 break;  
 }  
 }  
  
 if (poblacion != null) {  
 return poblacion.toString();  
 } else {  
 return "No se encontró la población con el nombre dado.";  
 }  
 }  
  
 */\*\*  
 \* Prints the names of all populations involved in the experiment.  
 \*/* public void imprimirNombresPoblaciones() {  
 for (Poblacion poblacion : this.poblaciones) {  
 System.*out*.println(poblacion.getNombre());  
 }  
 }  
  
 */\*\*  
 \* Edits a population based on the user's choice and the new parameters provided.  
 \* @param experimento The experiment containing the population to be edited  
 \* @param choice The index of the population to be edited  
 \* @param newName The new name for the population  
 \* @param newStartDateStr The new start date for the population  
 \* @param newEndDateStr The new end date for the population  
 \* @param newNumBacteria The new number of bacteria in the population  
 \* @param newTemperature The new temperature of the population  
 \* @param newLuminosityStr The new luminosity of the population  
 \* @param newDosis The new food dose of the population  
 \* @return A string indicating the result of the operation  
 \*/* public String editarPoblacion(Experimento experimento, int choice, String newName, String newStartDateStr, String newEndDateStr, Integer newNumBacteria, Float newTemperature, String newLuminosityStr, Dosis newDosis) {  
 StringBuilder output = new StringBuilder();  
  
 if (choice > 0 && choice <= experimento.poblaciones.size()) {  
 Poblacion poblacion = experimento.poblaciones.get(choice - 1);  
  
 if (newName != null) {  
 poblacion.setNombre(newName);  
 }  
  
 if (newStartDateStr != null) {  
 try {  
 SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");  
 Date newStartDate = formatter.parse(newStartDateStr);  
 poblacion.setFechaInicio(newStartDate);  
 } catch (ParseException e) {  
 output.append("Invalid date format. Please enter the date in the format yyyy-MM-dd.\n");  
 }  
 }  
  
 if (newEndDateStr != null) {  
 try {  
 SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");  
 Date newEndDate = formatter.parse(newEndDateStr);  
 poblacion.setFechaFin(newEndDate);  
 } catch (ParseException e) {  
 output.append("Invalid date format. Please enter the date in the format yyyy-MM-dd.\n");  
 }  
 }  
  
 if (newNumBacteria != null) {  
 poblacion.setNumBacterias(newNumBacteria);  
 }  
  
 if (newTemperature != null) {  
 poblacion.setTemperatura(newTemperature);  
 }  
  
 if (newLuminosityStr != null) {  
 Luminosidad newLuminosity = Luminosidad.*valueOf*(newLuminosityStr.toUpperCase());  
 poblacion.setLuminosidad(newLuminosity);  
 }  
  
 if (newDosis != null) {  
 poblacion.setDosisComida(newDosis);  
 }  
  
 } else {  
 output.append("Invalid choice. Please enter a number between 1 and " + poblaciones.size() + "\n");  
 }  
  
 return output.toString();  
 }  
  
 */\*\*  
 \* Returns a string representation of the Experimento.  
 \* @return A string representation of the Experimento  
 \*/* @Override  
 public String toString() {  
 return "Experimento{" +  
 "idExperimento=" + idExperimento +  
 ", nombreExp='" + nombreExp + '\'' +  
 '}';  
 }  
  
 *// Getters and setters for the Experimento's fields* public String getPathArchivo() {  
 return pathArchivo;  
 }  
  
 public void setPathArchivo(String pathArchivo) {  
 this.pathArchivo = pathArchivo;  
 }  
  
 public int getIdExperimento() {  
 return idExperimento;  
 }  
  
 public void setIdExperimento(int idExperimento) {  
 this.idExperimento = idExperimento;  
 }  
  
 public String getNombreExp() {  
 return nombreExp;  
 }  
  
 public void setNombreExp(String nombreExp) {  
 this.nombreExp = nombreExp;  
 }  
  
 public ArrayList<Poblacion> getPoblaciones() {  
 return poblaciones;  
 }  
  
 public void setPoblaciones(ArrayList<Poblacion> poblaciones) {  
 this.poblaciones = poblaciones;  
 }  
}

* 1. *Poblacion*

*/\*\*  
 \* The Poblacion class represents a population in an experiment.  
 \* It contains information about the population such as its name, id, start and end dates, number of bacteria, temperature, luminosity, and food dose.  
 \*/*public class Poblacion {  
 *// The name of the population* public String nombre;  
 *// The id of the population* public int idPoblacion;  
 *// The start date of the population* public Date fechaInicio;  
 *// The end date of the population* public Date fechaFin;  
 *// The number of bacteria in the population. It's an Integer because it needs to be null in the editPoblacion method* public Integer numBacterias;  
 *// The temperature of the population* public float temperatura;  
 *// The luminosity of the population* public Luminosidad luminosidad;  
 *// The food dose of the population* public Dosis dosisComida;  
  
 */\*\*  
 \* Constructs a new Poblacion with the given parameters.  
 \* @param nombre The name of the population  
 \* @param idPoblacion The id of the population  
 \* @param fechaInicio The start date of the population  
 \* @param fechaFin The end date of the population  
 \* @param numBacterias The number of bacteria in the population  
 \* @param temperatura The temperature of the population  
 \* @param luminosidad The luminosity of the population  
 \* @param dosisComida The food dose of the population  
 \*/* public Poblacion(String nombre, int idPoblacion, Date fechaInicio, Date fechaFin, int numBacterias,  
 float temperatura, Luminosidad luminosidad, Dosis dosisComida) {  
 this.nombre = nombre;  
 this.idPoblacion = idPoblacion;  
 this.fechaInicio = fechaInicio;  
 this.fechaFin = fechaFin;  
 this.numBacterias = numBacterias;  
 this.temperatura = temperatura;  
 this.luminosidad = luminosidad;  
 this.dosisComida = dosisComida;  
 }  
  
 */\*\*  
 \* Returns a string representation of the Poblacion.  
 \* @return A string representation of the Poblacion  
 \*/* @Override  
 public String toString() {  
 return "Poblacion{" +  
 "nombre='" + nombre + '\'' +  
 ", fechaInicio=" + fechaInicio +  
 ", fechaFin=" + fechaFin +  
 ", numBacteriasIniciales=" + numBacterias +  
 ", temperatura=" + temperatura +  
 ", luminosidad='" + luminosidad + '\'' +  
 ", dosisComida=" + dosisComida +  
 '}';  
 }  
  
 */\*\*  
 \* Checks if this Poblacion is equal to another object.  
 \* @param o The object to compare this Poblacion to  
 \* @return true if the objects are the same; false otherwise  
 \*/* @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (!(o instanceof Poblacion)) return false;  
 Poblacion poblacion = (Poblacion) o;  
 return idPoblacion == poblacion.idPoblacion && getNumBacterias() == poblacion.getNumBacterias() && Float.*compare*(getTemperatura(), poblacion.getTemperatura()) == 0 && Objects.*equals*(getNombre(), poblacion.getNombre()) && Objects.*equals*(getFechaInicio(), poblacion.getFechaInicio()) && Objects.*equals*(getFechaFin(), poblacion.getFechaFin()) && getLuminosidad() == poblacion.getLuminosidad() && Objects.*equals*(getDosisComida(), poblacion.getDosisComida());  
 }  
  
 */\*\*  
 \* Returns a hash code value for the Poblacion.  
 \* @return A hash code value for the Poblacion  
 \*/* @Override  
 public int hashCode() {  
 return Objects.*hash*(getNombre(), idPoblacion, getFechaInicio(), getFechaFin(), getNumBacterias(), getTemperatura(), getLuminosidad(), getDosisComida());  
 }  
  
 *// Getters and setters for the Poblacion's fields* public int getIdPoblacion() {  
 return idPoblacion;  
 }  
  
 public void setIdPoblacion(int idPoblacion) {  
 this.idPoblacion = idPoblacion;  
 }  
  
 public void setNombre(String nombre) {  
 this.nombre = nombre;  
 }  
  
 public void setFechaInicio(Date fechaInicio) {  
 this.fechaInicio = fechaInicio;  
 }  
  
 public void setFechaFin(Date fechaFin) {  
 this.fechaFin = fechaFin;  
 }  
  
 public void setNumBacterias(int numBacterias) {  
 this.numBacterias = numBacterias;  
 }  
  
 public void setTemperatura(float temperatura) {  
 this.temperatura = temperatura;  
 }  
  
 public void setLuminosidad(Luminosidad luminosidad) {  
 this.luminosidad = luminosidad;  
 }  
  
 public void setDosisComida(Dosis dosisComida) {  
 this.dosisComida = dosisComida;  
 }  
  
 public String getNombre() {  
 return nombre;  
 }  
  
 public Date getFechaInicio() {  
 return fechaInicio;  
 }  
  
 public Date getFechaFin() {  
 return fechaFin;  
 }  
  
 public int getNumBacterias() {  
 return numBacterias;  
 }  
  
 public float getTemperatura() {  
 return temperatura;  
 }  
  
 public Luminosidad getLuminosidad() {  
 return luminosidad;  
 }  
  
 public Dosis getDosisComida() {  
 return dosisComida;  
 }  
}

* 1. *Dosis*

*/\*\*  
 \* The Dosis class represents a dose in an experiment.  
 \* It contains information about the dose such as its initial amount, increment day, increment amount, and final amount.  
 \*/*public class Dosis {  
 *// The initial amount of the dose* double cantidadInicial;  
 *// The day on which the dose is incremented* int diaDeIncremento;  
 *// The amount by which the dose is incremented* double cantidadIncremento;  
 *// The final amount of the dose* double cantidadFinal;  
  
 */\*\*  
 \* Constructs a new Dosis with the given parameters.  
 \* @param cantidadInicial The initial amount of the dose  
 \* @param diaDeIncremento The day on which the dose is incremented  
 \* @param cantidadIncremento The amount by which the dose is incremented  
 \* @param cantidadFinal The final amount of the dose  
 \* @throws IllegalArgumentException If the increment day is 1 or 30, or if the initial or final amount exceeds 300, or if the increment amount exceeds 300 minus the initial amount  
 \*/* public Dosis(double cantidadInicial, int diaDeIncremento, double cantidadIncremento, double cantidadFinal){  
 if (diaDeIncremento == 1 || diaDeIncremento == 30) {  
 throw new IllegalArgumentException("Invalid increment day");  
 }  
 if (cantidadInicial > 300 || cantidadFinal > 300) {  
 throw new IllegalArgumentException("cantidadInicial and cantidadFinal must not exceed 300");  
 }  
 if (cantidadIncremento > (300 - cantidadInicial)) {  
 throw new IllegalArgumentException("cantidadIncremento must not exceed 300 - cantidadInicial");  
 }  
 this.cantidadInicial = cantidadInicial;  
 this.diaDeIncremento = diaDeIncremento;  
 this.cantidadIncremento = cantidadIncremento;  
 this.cantidadFinal = cantidadFinal;  
 }  
  
 */\*\*  
 \* Calculates the daily dose for a given Dosis object.  
 \* @param newdosis The Dosis object for which the daily dose is calculated  
 \*/* public void calcularDosisDiaria (Dosis newdosis){  
 double comidaDia = cantidadInicial;  
 double incrementoDiario = newdosis.getCantidadIncremento()/ (newdosis.getDiaDeIncremento()-1);  
 double decrementoDiario = (newdosis.getCantidadIncremento() + newdosis.getCantidadInicial() - newdosis.getCantidadFinal())/ (30 - getDiaDeIncremento());  
  
 for (int dia = 2; dia <= 30; dia++) {  
 if (dia <= newdosis.getDiaDeIncremento()) {  
 comidaDia = comidaDia + incrementoDiario;  
 System.*out*.println("Día " + dia + ": Se deben proporcionar " + comidaDia + " unidades de comida.");  
 } else {  
 comidaDia = comidaDia - decrementoDiario;  
 System.*out*.println("Día " + dia + ": Se deben proporcionar " + comidaDia + " unidades de comida.");  
 }  
 }  
 }  
  
 */\*\*  
 \* Returns a string representation of the Dosis.  
 \* @return A string representation of the Dosis  
 \*/* @Override  
 public String toString() {  
 return "Dosis{" +  
 " cantidadInicial=" + cantidadInicial +  
 ", diaDeIncremento=" + diaDeIncremento +  
 ", cantidadIncremento=" + cantidadIncremento +  
 ", cantidadFinal=" + cantidadFinal +  
 '}';  
 }  
  
 *// Getters and setters for the Dosis's fields* public double getCantidadInicial() {  
 return cantidadInicial;  
 }  
  
 public void setCantidadInicial(double cantidadInicial) {  
 this.cantidadInicial = cantidadInicial;  
 }  
  
 public int getDiaDeIncremento() {  
 return diaDeIncremento;  
 }  
  
 public void setDiaDeIncremento(int diaDeIncremento) {  
 this.diaDeIncremento = diaDeIncremento;  
 }  
  
 public double getCantidadIncremento() {  
 return cantidadIncremento;  
 }  
  
 public void setCantidadIncremento(double cantidadIncremento) {  
 this.cantidadIncremento = cantidadIncremento;  
 }  
  
 public double getCantidadFinal() {  
 return cantidadFinal;  
 }  
  
 public void setCantidadFinal(double cantidadFinal) {  
 this.cantidadFinal = cantidadFinal;  
 }  
}

* 1. *Luminosidad*

*/\*\*  
 \* The Luminosidad enum represents the level of luminosity.  
 \* It can be ALTA (high), MEDIA (medium), or BAJA (low).  
 \*/*public enum Luminosidad {  
 *ALTA*, *// High luminosity  
 MEDIA*, *// Medium luminosity  
 BAJA*; *// Low luminosity*}

1. *Paquete Tests*
   1. *TestsLaboratorio*

*/\*\*  
 \* This class contains test cases for the Laboratorio class.  
 \* It tests the functionality of creating, editing, and deleting Experimento objects in a Laboratorio.  
 \*/*public class TestsLaboratorio {  
 */\*\*  
 \* The Laboratorio object that will be used in the test cases.  
 \*/* private Laboratorio laboratorio;  
  
 */\*\*  
 \* Sets up the Laboratorio object before each test.  
 \*/* @BeforeEach  
 void setUp() {  
 this.laboratorio = new Laboratorio("Test Lab");  
 }  
  
 */\*\*  
 \* Test case for the method crearExperimento in the Laboratorio class.  
 \* This test case verifies that a new experiment is correctly added to the laboratory.  
 \* The test case follows these steps:  
 \* 1. Calls the method crearExperimento with valid parameters.  
 \* 2. Checks that the size of the experimentos list in the Laboratorio object has increased by 1.  
 \*/* @Test  
 void shouldAddExperimentoWhenCreatingNewOne() {  
 laboratorio.crearExperimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", new ArrayList<>());  
 *assertEquals*(1, laboratorio.experimentos.size());  
 }  
  
 */\*\*  
 \* Test case for the method editarExperimento in the Laboratorio class.  
 \* This test case verifies that the experiment is not changed when an invalid choice is provided.  
 \* The test case follows these steps:  
 \* 1. Calls the method crearExperimento to create a new experiment.  
 \* 2. Calls the method editarExperimento with an invalid choice.  
 \* 3. Checks that the experiment's id and name have not changed.  
 \*/* @Test  
 void shouldNotChangeExperimentoWhenEditingWithInvalidChoice() {  
 laboratorio.crearExperimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", new ArrayList<>());  
 String result = laboratorio.editarExperimento(2, "2", "Experimento 2");  
 *assertTrue*(result.contains("Invalid choice"));  
 *assertEquals*(1, laboratorio.experimentos.get(0).getIdExperimento());  
 *assertEquals*("Experimento 1", laboratorio.experimentos.get(0).getNombreExp());  
 }  
  
 */\*\*  
 \* Test case for the method editarExperimento in the Laboratorio class.  
 \* This test case verifies that the experiment is correctly changed when a valid choice is provided.  
 \* The test case follows these steps:  
 \* 1. Calls the method crearExperimento to create a new experiment.  
 \* 2. Calls the method editarExperimento with a valid choice.  
 \* 3. Checks that the experiment's id and name have been updated.  
 \*/* @Test  
 void shouldChangeExperimentoWhenEditingWithValidChoice() {  
 laboratorio.crearExperimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", new ArrayList<>());  
 laboratorio.editarExperimento(1, "2", "Experimento 2");  
 *assertEquals*(2, laboratorio.experimentos.get(0).getIdExperimento());  
 *assertEquals*("Experimento 2", laboratorio.experimentos.get(0).getNombreExp());  
 }  
  
 */\*\*  
 \* Test case for the method eliminarExperimentoPorId in the Laboratorio class.  
 \* This test case verifies that the experiment is not removed when an invalid id is provided.  
 \* The test case follows these steps:  
 \* 1. Calls the method crearExperimento to create a new experiment.  
 \* 2. Calls the method eliminarExperimentoPorId with an invalid id.  
 \* 3. Checks that the size of the experimentos list in the Laboratorio object has not changed.  
 \*/* @Test  
 void shouldNotRemoveExperimentoWhenDeletingWithInvalidId() {  
 laboratorio.crearExperimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", new ArrayList<>());  
 *assertFalse*(laboratorio.eliminarExperimentoPorId(2));  
 *assertEquals*(1, laboratorio.experimentos.size());  
 }  
  
 */\*\*  
 \* Test case for the method eliminarExperimentoPorId in the Laboratorio class.  
 \* This test case verifies that the experiment is correctly removed when a valid id is provided.  
 \* The test case follows these steps:  
 \* 1. Calls the method crearExperimento to create a new experiment.  
 \* 2. Calls the method eliminarExperimentoPorId with a valid id.  
 \* 3. Checks that the size of the experimentos list in the Laboratorio object has decreased by 1.  
 \*/* @Test  
 void shouldRemoveExperimentoWhenDeletingWithValidId() {  
 laboratorio.crearExperimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", new ArrayList<>());  
 *assertTrue*(laboratorio.eliminarExperimentoPorId(1));  
 *assertEquals*(0, laboratorio.experimentos.size());  
 }  
}

* 1. *TestsExperimento*

*/\*\*  
 \* This class contains test cases for the Experimento class.  
 \* It tests the functionality of adding, removing, creating, and editing Poblacion objects in an Experimento.  
 \*/*class TestExperimento {  
 private Experimento experimento;  
 private Poblacion poblacion1;  
 private Poblacion poblacion2;  
  
 */\*\*  
 \* This method sets up the common objects used in the test cases.  
 \* It is annotated with @BeforeEach, so it runs before each test case.  
 \*/* @BeforeEach  
 void setUp() {  
 this.experimento = new Experimento(1, "Experimento 1", "Some String", new ArrayList<>());  
 poblacion1 = new Poblacion("Poblacion1", 1, new Date(), new Date(), 100, 37.0f, Luminosidad.*ALTA*, new Dosis(150,15,130,150));  
 poblacion2 = new Poblacion("Poblacion2", 2, new Date(), new Date(), 200, 37.0f, Luminosidad.*BAJA*, new Dosis(100,15,100,100));  
 }  
  
 */\*\*  
 \* This test case checks if the agregarPoblacion method correctly adds a Poblacion to an Experimento.  
 \*/* @Test  
 void shouldAddPoblacion() {  
 experimento.agregarPoblacion(poblacion1);  
 *assertTrue*(experimento.getPoblaciones().contains(poblacion1));  
 }  
  
 */\*\*  
 \* This test case checks if the agregarPoblacion method does not add a null Poblacion to an Experimento.  
 \*/* @Test  
 void shouldNotAddNullPoblacion() {  
 experimento.agregarPoblacion(null);  
 *assertFalse*(experimento.getPoblaciones().contains(null));  
 }  
  
 */\*\*  
 \* This test case checks if the borrarPoblacion method correctly removes a Poblacion from an Experimento.  
 \*/* @Test  
 void shouldRemovePoblacion() {  
 experimento.agregarPoblacion(poblacion1);  
 experimento.borrarPoblacion(poblacion1);  
 *assertFalse*(experimento.getPoblaciones().contains(poblacion1));  
 }  
  
 */\*\*  
 \* This test case checks if the borrarPoblacion method does not remove a non-existent Poblacion from an Experimento.  
 \*/* @Test  
 void shouldNotRemoveNonExistentPoblacion() {  
 experimento.agregarPoblacion(poblacion1);  
 experimento.borrarPoblacion(poblacion2);  
 *assertTrue*(experimento.getPoblaciones().contains(poblacion1));  
 }  
  
 */\*\*  
 \* This test case checks if the crearPoblacion method correctly creates a Poblacion and adds it to an Experimento.  
 \*/* @Test  
 void shouldCreatePoblacion() {  
 experimento.crearPoblacion("Poblacion3", 3, new Date(), new Date(), 300, 37.0f, Luminosidad.*MEDIA*, new Dosis(160,15,100,100));  
 *assertEquals*(1, experimento.getPoblaciones().size());  
 *assertEquals*("Poblacion3", experimento.getPoblaciones().get(0).getNombre());  
 }  
  
 */\*\*  
 \* This test case checks if the editarPoblacion method correctly edits a Poblacion in an Experimento.  
 \*/* @Test  
 void shouldEditPoblacion() {  
 *// Set up the Experimento and Poblacion* Experimento experimento = new Experimento(1, "Experimento 1", "Some String", new ArrayList<>());  
 Poblacion poblacion = new Poblacion("Poblacion1", 1, new Date(), new Date(), 100, 37.0f, Luminosidad.*ALTA*, new Dosis(150,15,130,150));  
 experimento.agregarPoblacion(poblacion);  
  
 *// Call editarPoblacion with the new parameters* String output = experimento.editarPoblacion(experimento, 1, "NewName", "2022-12-31", "2023-12-31", null, null, null, null);  
  
 *// Check that the Poblacion has been updated correctly  
 assertEquals*("NewName", poblacion.getNombre());  
 SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");  
 try {  
 *assertEquals*(formatter.parse("2022-12-31"), poblacion.getFechaInicio());  
 *assertEquals*(formatter.parse("2023-12-31"), poblacion.getFechaFin());  
 } catch (ParseException e) {  
 *fail*("Failed to parse date");  
 }  
  
 *// Check the output* String expectedOutput = "";  
 *assertEquals*(expectedOutput, output);  
 }  
  
 */\*\*  
 \* This test case checks if the mostrarPoblacion method correctly returns a string representation of a Poblacion in an Experimento.  
 \*/* void mostrarPoblacion() {  
 *// Create a new Poblacion* Poblacion poblacion = new Poblacion("Test Poblacion", 1, new Date(), new Date(), 100, 37.0f, Luminosidad.*ALTA*, new Dosis(100, 15, 200, 100));  
  
 *// Create a new Experimento and add it to the Laboratorio* Experimento experimento = new Experimento(1, "Test Experimento", "path/to/file", new ArrayList<>());  
 Laboratorio laboratorio = new Laboratorio("Test Lab");  
 laboratorio.getExperimentos().add(experimento);  
  
 *// Add the Poblacion to the Experimento* experimento.getPoblaciones().add(poblacion);  
  
 *// Call the method we want to test* String result = experimento.mostrarPoblacion("Test Poblacion");  
  
 *// Check the result  
 assertEquals*("Poblacion{" +  
 "nombre='Test Poblacion'" +  
 ", fechaInicio=" + poblacion.getFechaInicio() +  
 ", fechaFin=" + poblacion.getFechaFin() +  
 ", numBacteriasIniciales=" + poblacion.getNumBacterias() +  
 ", temperatura=" + poblacion.getTemperatura() +  
 ", luminosidad='" + poblacion.getLuminosidad() + '\'' +  
 ", dosisComida=" + poblacion.getDosisComida() +  
 '}', result);  
 }  
}

* 1. *TestsGestionDeMemoria*

*/\*\*  
 \* This class contains test cases for the GestionDeMemoria module.  
 \* It tests the functionality of saving and loading Experimento objects to and from files.  
 \*/*class TestsGestionDeMemoria {  
 private Experimento experimento;  
 private Poblacion poblacion1;  
 private Poblacion poblacion2;  
  
 */\*\*  
 \* This method sets up the common objects used in the test cases.  
 \* It is annotated with @BeforeEach, so it runs before each test case.  
 \*/* @BeforeEach  
 void setUp() {  
 this.experimento = new Experimento(1, "Experimento 1", "Some String", new ArrayList<>());  
 poblacion1 = new Poblacion("Poblacion1", 1, new Date(), new Date(), 100, 37.0f, Luminosidad.*ALTA*, new Dosis(150, 15, 130, 150));  
 poblacion2 = new Poblacion("Poblacion2", 2, new Date(), new Date(), 200, 37.0f, Luminosidad.*BAJA*, new Dosis(100, 15, 150, 100));  
 }  
  
 */\*\*  
 \* This test case checks if the guardarExperimentoComo method correctly saves an Experimento object to a file.  
 \* It creates a temporary file, an Experimento object, and a Poblacion object, and adds the Poblacion to the Experimento.  
 \* It then calls guardarExperimentoComo and checks if the contents of the file match the details of the Experimento and its Poblacion.  
 \*/* @Test  
 void shouldSaveExperimentToFile() throws IOException {  
 *// Set up the Experimento and Poblacion* File tempFile = File.*createTempFile*("tempFile", ".txt");  
 Experimento experimento = new Experimento(1, "Experimento 1", tempFile.getAbsolutePath(), new ArrayList<>());  
 Poblacion poblacion = new Poblacion("Poblacion1", 1, new Date(), new Date(), 100, 37.0f, Luminosidad.*ALTA*, new Dosis(150, 15, 130, 150));  
 experimento.agregarPoblacion(poblacion);  
  
 *// Call guardarExperimentoEnArchivo* GuardarEnMemoria.*guardarExperimentoComo*(experimento);  
  
 *// Open the file and read its contents* try {  
 File file = new File(experimento.getPathArchivo());  
 Scanner scanner = new Scanner(file);  
 StringBuilder fileContents = new StringBuilder();  
 while (scanner.hasNextLine()) {  
 fileContents.append(scanner.nextLine()).append("\n");  
 }  
 scanner.close();  
  
 *// Check that the contents of the file match the details of the Experimento and its Poblacion* String expectedOutput = "Experimento ID: " + experimento.getIdExperimento() + "\n" +  
 "Experimento Name: " + experimento.getNombreExp() + "\n" +  
 "\n" +  
 "Poblacion ID: " + poblacion.getIdPoblacion() + "\n" +  
 "Poblacion Name: " + poblacion.getNombre() + "\n" +  
 "Start Date: " + poblacion.getFechaInicio() + "\n" +  
 "End Date: " + poblacion.getFechaFin() + "\n" +  
 "Number of Bacteria: " + poblacion.getNumBacterias() + "\n" +  
 "Temperature: " + poblacion.getTemperatura() + "\n" +  
 "Luminosity: " + poblacion.getLuminosidad() + "\n" +  
 "Food Dose: " + poblacion.getDosisComida() + "\n";  
 *assertEquals*(expectedOutput, fileContents.toString());  
 } catch (FileNotFoundException e) {  
 *fail*("Failed to open the file");  
 }  
 }  
  
 */\*\*  
 \* This test case checks if the leerExperimentoDesdeArchivo method correctly loads an Experimento object from a valid file.  
 \* It creates an Experimento object and calls leerExperimentoDesdeArchivo with a valid file path.  
 \* It then checks if the loaded Experimento's id and name match the expected values.  
 \*/* @Test  
 void shouldCreateExperimentoFromValidFile() {  
 Experimento experimento = new Experimento(0, null, "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt", null);  
 Experimento loadedExperimento = LeerDeMemoria.*leerExperimentoDesdeArchivo*("C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimento1.txt");  
 *assertNotNull*(loadedExperimento);  
 *assertEquals*(1, loadedExperimento.getIdExperimento());  
 *assertEquals*("Experimento 1", loadedExperimento.getNombreExp());  
 }  
  
 */\*\*  
 \* This test case checks if the leerExperimentoDesdeArchivo method throws an IllegalArgumentException when given an invalid file format.  
 \* It creates an Experimento object and calls leerExperimentoDesdeArchivo with an invalid file path.  
 \* It then checks if an IllegalArgumentException is thrown.  
 \*/* @Test  
 void shouldNotCreateExperimentoFromInvalidFileFormat() {  
 Experimento experimento = new Experimento(0, null, "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimentoNOvalido.txt", null);  
 *assertThrows*(IllegalArgumentException.class, () -> LeerDeMemoria.*leerExperimentoDesdeArchivo*("C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimentoNOvalido.txt"));  
 }  
  
 */\*\*  
 \* This test case checks if the guardar method does not throw an exception when given a valid Experimento object.  
 \* It creates an Experimento object and calls guardar.  
 \* It then checks if no exception is thrown.  
 \*/* @Test  
 void shouldSaveExperimentoToValidFile() {  
 Experimento experimento = new Experimento(1, "Experimento 1", "C:\\Users\\pipe\\Documents\\Programacion CEU\\Java\\IntelliJ\\Segundo Curso\\Archivos de Texto Practica 1\\experimentoNOvalido.txt", new ArrayList<>());  
 *assertDoesNotThrow*(() -> GuardarEnMemoria.*guardar*(experimento));  
 }  
  
 */\*\*  
 \* This test case checks if the guardar and leerExperimentoDesdeArchivo methods work correctly together.  
 \* It creates an Experimento object, calls guardar, and then calls leerExperimentoDesdeArchivo.  
 \* It then checks if the loaded Experimento's id and name match the original Experimento's id and name.  
 \*/* @Test  
 void shouldSaveAndLoadExperimentoCorrectly() {  
 Experimento experimento = new Experimento(1, "Experimento 1", "validFilePath", new ArrayList<>());  
 GuardarEnMemoria.*guardar*(experimento);  
 Experimento loadedExperimento = LeerDeMemoria.*leerExperimentoDesdeArchivo*("validFilePath");  
 *assertEquals*(experimento.getIdExperimento(), loadedExperimento.getIdExperimento());  
 *assertEquals*(experimento.getNombreExp(), loadedExperimento.getNombreExp());  
 }  
}

* 1. *TestDosis*

*/\*\*  
 \* This class contains a test for the Dosis class.  
 \* It tests the functionality of creating a Dosis object and calculating the daily dose.  
 \*/*public class TestDosis {  
 *// BufferedReader to read input from the console* public static BufferedReader *bufr* = new BufferedReader(new InputStreamReader(System.*in*));  
  
 */\*\*  
 \* The main method which is the entry point for the test.  
 \* It prompts the user to enter the parameters for a Dosis object, creates the object, and calculates the daily dose.  
 \* @param args Command line arguments. Not used in this method.  
 \* @throws IOException If an input or output exception occurred  
 \*/* public static void main(String[] args) throws IOException {  
 try {  
 *// Prompt the user to enter the parameters for a Dosis object* System.*out*.println("Creando dosis");  
 System.*out*.println("introduzca la cantidad inicial: ");  
 double cantidadInicial = Integer.*parseInt*(*bufr*.readLine());  
 System.*out*.println("introduzca el dia de incremento: ");  
 int diaDeIncremento = Integer.*parseInt*(*bufr*.readLine());  
 System.*out*.println("introduzca la cantidad de incremento: ");  
 double cantidadIncremento = Integer.*parseInt*(*bufr*.readLine());  
 System.*out*.println("introduzca la cantidad final: ");  
 double cantidadFinal = Integer.*parseInt*(*bufr*.readLine());  
  
 *// Create a Dosis object with the entered parameters* Dosis dosistest = new Dosis(cantidadInicial, diaDeIncremento, cantidadIncremento, cantidadFinal);  
  
 *// Calculate the daily dose* dosistest.calcularDosisDiaria(dosistest);  
  
 } catch (IOException e) {  
 *// Print an error message if an input or output exception occurred* System.*out*.println("Error en la entrada de datos");  
 }  
 }  
}