**ACTIVIDAD 2 - INDIVIDUAL - POO**

**OBSERVACIONES IMPORTANTE ANTES DE INICIAR:**

Cada trabajo o actividad académica que sea desarrollado por usted como evidencia de su proceso de aprendizaje, debe estar 100% libre de frases copiadas desde cualquier fuente original (documento, libro, página web, proyecto de código fuente, video tutorial, etc.) de información.

En ningún momento se le está pidiendo que haga las cosas de 0 y olvide por completo los conocimientos sobre los cuales se basa algún tema de esta asignatura, por el contrario, lo estamos animando a que lea, investigue, comprenda, deduzca, analice, sintetice, concluya, interprete, argumente, domine, practique, critique, niegue o afirme con bases sólidas basadas en la lectura y el estudio autónomo y autodidacta.

Tenga muy presente que mientras estudias esta profesión y durante toda tu vida, siempre seras evaluado y medido de acuerdo por un docente, por un instructor, por tu equipo de trabajo, por tus jefes, por la sociedad o por tus clientes.

Antes de realizar cualquier acción indebida en el desarrollo de esta y cualquier otra actividad académica, recuerda que actualmente existen muchas herramientas modernas y avanzadas que existen analizar, buscar, identificar, rastrear y comparar la idoneidad y originalidad de cada contenidos digital (texto, imágenes, archivos, etc), con especial profundidad en archivos de documentos académico o que contienen fuentes de programas informáticos, las cuales permiten detectar plagio de una manera tremendamente simple, rápida y automatizada.

Evita que el pensamiento del mínimo esfuerzo y el facilismo invada tu mente y tus acciones, ya que este destruirá tus sueños de crecimiento y éxito profesional.

**OBSERVACIONES**:

* **actividad para desarrollar de forma individual**

● Se debe subir al aula virtual de la asignatura por sima-pesad (no se aceptan entregas por otro medio)

● Solo se evaluará y calificará a los estudiantes que hayan subido la actividad por sima-pesad en la fecha estipulada

● Existen muchísimas formas (lógicas y estructurales) de desarrollar un mismo ejercicio de programación, **se evalúa con 1.0 cualquier evidencia de similitud del contenido del código o trabajo con otros contenidos entregados por los alumnos,** **encontrados por internet.** Recuerde que no es ilegal aprender de otros recursos, **lo ilegal es tomar ese recurso para aprovecharse del trabajo hecho por otra persona para luego maquillarlo para tratar de hacerlo pasar por nuestro y ganar una buena calificación,** es decir, **eviden realizar los siguientes trucos viejos y muy fáciles de evidenciar**:

Si la IA te resuelve el los ejercicios, o un amigo o compañero te pasa su tarea, o le pagas a alguien para que la haga por ti o la encuentras resuelta por internet, solo recuerda que en estos momentos ninguno de ellos está perdiendo el tiempo mas que tu, ni actúa de forma más absurda, ilógica e irracional que tu, al realizar tal acción.

**ESTA SON ALGUNAS DE EVIDENCIAS DE PLAGIO O FRAUDE MÁS COMUNES Y FÁCILES DE IDENTIFICAR SIN NECESIDAD DE USAR HERRAMIENTAS TECNOLÓGICAS:**

○ Reorganizar el código, colocando clases, variables y métodos en otros lugares diferentes al lugar donde los dejó el autor original.

○ Cambiar nombres de clases

○ Cambiar nombre de archivos

○ Cambiar nombre del proyecto

○ Cambiar nombres de variables

○ Cambiar el nombre de los métodos o funciones,

○ Cambiar mensajes de salida,

○ Cambiar System.out.print por JOptionPane

○ Cambiar color y apariencia de los componentes de GUI

○ etc

● **Recuerda que:**

* Estudiar y practicar en grupo (cipa) no es sinónimo de entregar el mismo el mismo resultado, que una persona desarrolle el ejercicio y se lo pase a los compañeros de grupo para que lo presenten cuando la tarea es individual.
* Cada estudiante debe tener la capacidad, habilidad, competencia y destreza para poder desarrollar todos los ejercicios de forma individual y con su propia lógica.

**La mejor forma de evitar realizar fraude académico es:**1. Estudiar frecuente y constantemente

2. Identificar prioridades en tu vida y en tu dia a dia

3. Construir un plan de trabajo colocando mayor peso (importancia) a los compromisos que tienen con mayor prioridad para tu vida

4. Ejecuta ese plan dia a dia, evitando posponer, retrasar o aplazar las actividades y compromisos diarios

5. Utiliza internet y sus poderosas herramientas para buscar recursos académicos de los cuales puedas aprender y practicar (Buscadores de recursos académicos, Foros, Libros digitales, cursos en video, tutoriales sobre temas puntuales, etc) y estudiar con base a sus contenidos.

6. Reúnete con tus compañeros (pueden ser de la U, o amigos de barrio, conocidos, etc, que tengan tu misma visión, alto grado de responsabilidad, tus mismas metas y objetivos), esto con el fin de intercambiar conocimientos mutuamente, tratando siempre de aportar y más que preguntar.

7. Pon en práctica los conocimientos adquiridos durante el proceso de estudio, para eso tienes las actividades, pero también puedes diseñar tus propias actividades personales de aprendizaje.

8. Autoevaluate con mucha frecuencia, de esta manera podrás saber como vas, que estas haciendo mal, que estás haciendo bien, si esta lejos o cerca del objetivo.

**FORMATO DE ENTREGA:**

Un archivo comprimido en formato ZIP **(NO USAR OTRO FORMATO) - NO SE ACEPTA**

**Nombre del archivo**:

codigo\_nombre\_apellido\_asignatura\_act\_2\_2023-2**.zip**

**Por ejemplo**:

1234567\_john\_arrieta\_poo\_act\_2\_2023\_2.**zip**

El archivo comprimido debe contener como mínimo los siguientes archivos.

■ Documento en formato EDITABLE .DOC (NO ES PDF) con el d**esarrollo y explicación detallada** del desarrollo de la actividad.

■ El desarrollo y **entrega del trabajo debe contener todos los ítem** típicos y característicos de un trabajo académico.

1. Presentación

2. Tabla de contenido

3. Introducción

4. Objetivos

5. Justificación

6. Desarrollo

a. Debe ser lo suficientemente explicativo,

b. Argumentativo

c. Ilustrativo

d. Enlace de GitHub del repositorio para descargar los proyectos o archivos de código fuente.

7. Bibliografía

Archivo comprimido con los códigos fuentes del desarrollo de la actividad.

**Normas APA para la elaboración de esta actividad**

<https://www.colconectada.com/normas-apa/>

**DESARROLLO DE LA ACTIVIDAD ACADÉMICA #3**

* Deben tomar fragmentos o recortes de pantallazos del código los cuales deben insertarlos en el documento como parte del desarrollo y respuesta de cada ítem descrito en esta actividad.
* Cada ítem debe contener su respectiva explicación detallada y conceptual, indicando que es, porque es y para qué es

**NO USAR IMÁGENES DE LA PANTALLA COMPLETA, SOLO RECORTES DE LA ZONA ESPECÍFICA**

Basados en el resultado obtenido con el desarrollo de la actividad individual entregada en la Unidad #1, utilizar el MODELO CONCEPTUAL (diagrama de clases UML realizado en la fase de análisis) y construir transformar el diagrama de clases a Clases en código fuente del lenguaje de programación Java.

MATERIAL DE APOYO.

[EJERCICIO DE POO COMPLETO CON CRUD CON LISTAS Y MAPAS - CURSO DE JAVA](https://www.youtube.com/watch?v=hv9_cyhKhbU)

ARREGLOS PARTE 1: <https://www.youtube.com/watch?v=7piqgA62oe0&t=6s>

ARREGLOS PARTE 2: <https://www.youtube.com/watch?v=Q9vxf3FUh3Y&t=12s>

ARREGLOS PARTE 2: <https://www.youtube.com/watch?v=655QdfG-VX8&t=8s>

ARREGLOS PARTE 3: <https://www.youtube.com/watch?v=BP6RtGBv5Wk&t=34s>

[CRUD DE OBJETOS USANDO LISTAS Y MAPAS JAVA](https://www.youtube.com/watch?v=R5WxUNEVP5M&t=3186s)

[Estructuras de Datos - ARREGLOS Ejemplo en Java parte 4 - Vista Agregar](https://www.youtube.com/watch?v=zOqNhlKbnN0)

[MisionTic Ciclo 2 - Java - Tutoria 50 - ArrayList - HashMap con Junit](https://www.youtube.com/watch?v=8lGSACLlyvY)

[MisionTic Ciclo 2 - Java - Tutoría 47 - ArrayList - HashMap y Junit](https://www.youtube.com/watch?v=lkLZGgetGrk)

[MisionTic Ciclo 2 - Java - Tutoría 47 - ArrayList - HashMap y Junit](https://www.youtube.com/watch?v=lkLZGgetGrk)

<https://www.youtube.com/watch?v=EiAs9mqjeg0>

<https://www.youtube.com/watch?v=-NWEgWJzB8o>

[MisionTic Ciclo 2 - Java - Tutoria 48 - ArrayList HashMap y Junit](https://www.youtube.com/watch?v=fF66-bdYPgQ)

[MisionTic Ciclo 2 - Java - Tutoria 50 - ArrayList - HashMap con Junit](https://www.youtube.com/watch?v=8lGSACLlyvY)

[MisionTic Ciclo 2 - Java - Tutoria 49 - ArrayList, HashMap con Junit](https://www.youtube.com/watch?v=rxBH1XxmEq4&t=46s)

[MisionTIC - CICLO 2 - JAVA - TUTORIA 22 - CADENAS STRING - ARREGLOS O ARRAY - FOREACH](https://www.youtube.com/watch?v=Lw7lPE56w-Y)

[MisionTIC - CICLO 2 - JAVA - TUTORIA 21 - CADENAS STRING - ARREGLOS O ARRAY - FOREACH](https://www.youtube.com/watch?v=ApjMaMk8iZ8)

**Actividades a desarrollar:**

Continuar con el proyecto desarrollado en la actividad 1 de acuerdo al ejercicio que les tocó.

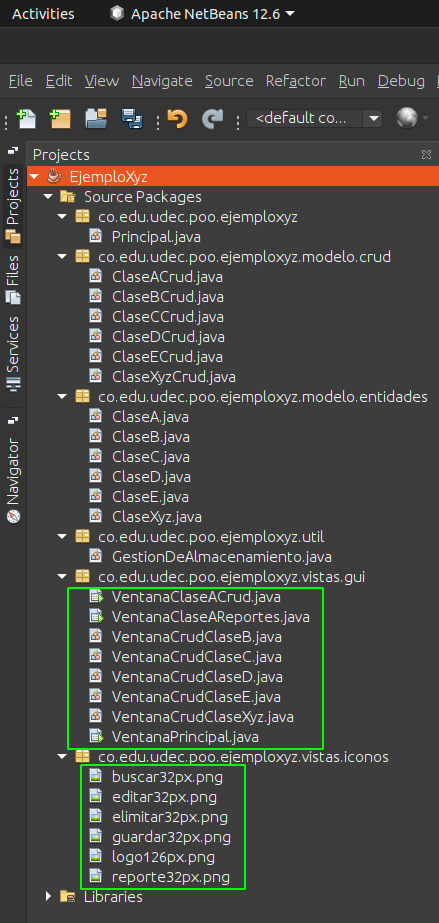
1. Crear una clase llamada clase **ClaseXyzCrud.java**, donde Xyz se refiere al nombre de cada una de las clases entidades del diagrama de clases desarrollado en el la actividad anterior.

**Ejemplo**: Si en el diagrama de clase existe una clase llamada **Producto**, entonces, debe existir una clase entidad Java llamada **Producto.java**, la cual debe tener las respectivas **propiedades**, métodos constructores, métodos **get** y métodos **set**, etc.

Tambien debe existir una clase llamada **ProductoCrud.java**

Cada **ClaseXyzCrud.java** debe tener una lista (**ArrayList** o **HashMap**) de tipo **ClaseXyz.**

La estructura de general del proyecto debe tener una apariencia similar a la siguiente imagen:



1. Cada clase **ClaseXyzCrud**.java debe tener los siguientres metodos:
   1. public void **agregar**(**ClaseXyz** **objeto**) throw **Exception**
   2. public **Abc buscar**(Integer **codigo**) throw **Exception**
   3. public void **editar**(**ClaseXyz** **objeto**) throw **Exception**
   4. public void **eliminar**(Integer **codigo**) throw **Exception**
   5. public **ArrayList**<**ClaseXyz**> **listarTodo(** ) throw **Exception**
   6. public Integer **contar**( ) throw **Exception**
   7. El método **agregar** debe:
      * Agregar en la lista (si no existe) el objeto **ClaseXyz** que recibe como parámetro de entrada
      * Si el objeto **ClaseXyz** existe previamente guardado en la lista, el método debe lanzar una excepción con un mensaje informativo
   8. El método **buscar** debe:
      * Usar el código que recibe como parámetro de entrada para buscar un objeto **ClaseXyz** en la lista, si lo encuentra lo retorna,
      * Si no lo encuentra debe lanzar una excepción con un mensaje informativo
   9. El método **editar** debe:
      * Cambiar en la lista (si existe) el objeto **ClaseXyz** que recibe como parámetro de entrada
      * Si el objeto **ClaseXyz** NO Existe en la lista, el método debe lanzar una excepción con un mensaje informativo
   10. El método **eliminar** debe:
       * Usar el código que recibe como parámetro de entrada para buscar un objeto **ClaseXyz** en la lista y borrarlos en caso de que este exista
       * Si no lo encuentra debe lanzar una excepción con un mensaje informativo
   11. El método **listarTodo** debe:
       * Devolver la lista de objetos **ClaseXyz**
       * Si la lista está vacía debe lanzar una excepción con un mensaje informativo
   12. El método **contar** debe:
       * Retornar el número de objetos **ClaseXyz** que existen dentro de la lista

Instrucción tomada como referente:

package co.edu.ude.poo.johnarrieta.ejercicioxyc.modelo.crud;

import java.util.\*;

import co.edu.ude.poo.johnarrieta.ejercicioxyc.modelo.entidades;

public class ClaseACrud {

private List<ClaseA> listaObjetosA;

// puede usar la clase Map

// private Map<String, ClaseA> diccionarioObjetosA;

public static void agregar(ClaseA objetoA) throw Exception{

// Usar los metodos correctos correctos de la clase List o clase Map

}

public static ClaseA buscar(Integer codigo) throw Exception{

// Usar los metodos correctos de la clase List o clase Map

}

public static void editar(ClaseA objetoA) throw Exception{

// Usar los metodos correctos de la clase List o clase Map

}

public static void eliminar(Integer codigo) throw Exception{

// Usar los metodos correctos de la clase List o clase Map

}

public static List<ClaseA> listarTodo( ) throw Exception{

// Usar los metodos correctos de la clase List o clase Map

}

public static Integer contar( ) throw Exception{

// Usar los metodos correctos de la clase List o clase Map

}

// ...

}

// continuer con las clases Crud para cada clase del paquete Entidades

1. En la clase **Principal** (la que tiene el método main) se debe:
   1. Crear al menos 2 objetos de cada clase entidad **ClaseXyz** (por ejemplo Producto, etc)

**ClaseXyz objeto1Xyz = new ClaseXyz();**

**ClaseXyz objeto2Xyz = new ClaseXyz();**

**ClaseXyz objeto3Xyz = new ClaseXyz();**

// ...

* 1. utilizar (invocar) los métodos set de cada objeto para pasarle datos a sus propiedades.

**objeto1Xyz**.setPropiedad1(parametro);

//...

* 1. Utilizar (invocar) los métodos **agregar** de cada objeto **ClaseXyzCrud** para agregar objetos **ClaseXyz** en su respectiva lista

**try{**

**//...**

**ClaseXyzCrud.agregar(objeto1Xyz);**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Invocar el método contar el número de objetos que existen en cada lista

**try{**

**//...**

**int total = ClaseXyzCrud.contar();**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Invocar el método buscar de cada objeto **ClaseXyzCrud** y pasarle un código para que busque ese objeto en la lista

**try{**

**//...**

**ClaseXyz xyz = ClaseXyzCrud.buscar(dato);**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Mostrar los datos del objeto encontrado, o el mensaje de error enviado por la excepción en caso de que no exista

**try{**

**//...**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Usar al menos dos objetos de tipo **ClaseXyz** para cambiar los datos de sus propiedades

**try{**

**//...**

**xyz.setPropiedad1(nuevoDato1);**

**xyz.setPropiedad2(nuevoDato2);**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. invocar el método editar para cambiar los datos de ese objeto la lista, o mostrar el mensaje de la excepción en caso de que no exista

**try{**

**//...**

**ClaseXyzCrud.editar(xyz);**

**// ...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Invocar el método eliminar del objeto **ClaseXyzCrud** para pasarle un código y tratar de eliminar ese objeto en la lista en caso de que exista, y mostrar el nuevo número de objetos en la lista, o mostrar el mensaje de la **Excepción** en caso de que no exista

**try{**

**//...**

**ClaseXyzCrud.eliminar(dato);**

**//...**

**}catch(Exception err){**

**// ...**

**}**

* 1. Invocar el método **listarTodo** para recorrer la lista de objetos **ClaseXyz** y mostrar en pantalla la información de cada uno

**try{**

**//...**

**List<ClaseXyz> listaXyz = ClaseXyzCrud.listar();**

**for( ClaseXyz xyz : listaXyz){**

**// ...**

**}**

**}catch(Exception err){**

**// ...**

**}**

12. Enunciar todos los contactos del lenguaje de programación Java que ha aprendido con el desarrollo de esta actividad, los cuales desconocía anteriormente, haga una tabla o una lista de ellos y explique con sus propias palabras para que utilizó dicho concepto en el desarrollo de la tarea.