



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- **Denominación del Programa de Formación:** Tecnólogo en Análisis y Desarrollo de Software.
- **Código del Programa de Formación:** 228118.
- **Nombre del Proyecto:** Construcción de software integrador de tecnologías orientadas a servicios
- **Fase del Proyecto:** Planeacion.
- **Actividad de Proyecto:** .
- **Competencia:** Determinar los requisitos funcionales y no funcionales del software de acuerdo con los requerimientos del cliente
- **Resultados de Aprendizaje Alcanzar:**
 - Modelar las funciones del software de acuerdo con el informe de requisitos.
- **Duración de la Guía:** XX horas

2. PRESENTACIÓN.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- **Descripción de la(s) Actividad(es)**

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Descripción de la actividad:

Identifica de forma individual los conceptos relacionados con los diagramas de caso de uso y mediante un documento relacionarlos teniendo en cuenta los siguientes interrogantes:

- Que es el lenguaje unificado de modelado “UML”
- Cuales son las Características de UML
- Que Tipos de diagramas UML y su uso.
- Que es el diagrama de Casos de uso.



- Que Componentes definen los casos de uso.
- Cual es su representación gráfica
- Que son y que Tipos de relaciones existen en la construcción de los casos de usos.
- Cuales son los parametros para la Documentación de un caso de uso.

Ambiente requerido: Equipos de computo, conexión a Internet

Estrategias o técnicas didácticas activas:

Materiales de formación:

Material de apoyo:

Duración de la actividad: 4 horas.

Evidencias de aprendizaje:

GA1—AA1- EV01-Reconocer los conceptos basicos el lenguaje de modelado unificado UML y la construcción de los artefactos de Casos de uso.

El documento debe contener normas APA y la siguiente estructura:

- Presentación (Portada)
- Introducción
- Desarrollo del trabajo
- Conclusiones

Se debe realizar una **Infografía** sobre los siguientes puntos, con ejemplos.

- Que Tipos de diagramas UML y su uso.
- Que es el diagrama de Casos de uso.
- Que Componentes definen los casos de uso.
- Cual es su representación gráfica
- Que son y que Tipos de relaciones existen en la construcción de los casos de usos.
- Cuales son los parametros para la Documentación de un caso de uso.
- **Cuestionario Técnico.** Con el fin de obtener evidencia de esta actividad de apropiación, con relación a los conocimientos adquiridos por el aprendiz para el logro de los resultados de aprendizaje previstos.

3.3 Actividades de apropiación:

3.4 Actividades de Transferencia el Conocimiento:

Descripción de la actividad:



Ambiente requerido:

Estrategias o técnicas didácticas activas:

Materiales de formación

Material de apoyo:

Evidencias de aprendizaje:

Instrumentos de evaluación:

Duración de la actividad: horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación

5. GLOSARIO DE TÉRMINOS

- **Encuesta:** Conjunto de preguntas tipificadas dirigidas a una muestra representativa, para averiguar estados de opinión o diversas cuestiones de hecho.
- **Entrevista:** Conferencia entre dos o más personas en lugar determinado, sustentada en una lista de preguntas estructuradas con el objetivo de obtener información.
- **Entrevistado:** Persona a quien va dirigida la entrevista, es quien responde el cuestionario.
- **Entrevistador:** Persona encargada de preparar, aplicar y analizar la entrevista.
- **Estructura:** Distribución adecuada de un grupo de preguntas para ser aplicadas en una entrevista.
- **Categorización:** consiste en la recolección y clasificación de la información de manera ordenada.
- **Investigación:** Su fin es ampliar el conocimiento científico, sin perseguir, en principio, ninguna aplicación práctica.
- **Método:** Modo en que se procede y los procedimientos seguidos para hallar la verdad acerca del objeto de investigación.
- **Metodología:** ciencia que consta de métodos y técnicas, que se aplican sistemáticamente durante un proceso de investigación o para solucionar una problemática.
- **Población:** Conjunto de elementos o personas sobre los que se está interesado en realizar una observación para obtener información relevante para el sistema de información.
-



6. REFERENTES BIBLIOGRÁFICOS

- Gómez, M. (2006). *Introducción a la metodología de la investigación científica*. Córdoba, Argentina: Brujas.
- Kendall, K. &. (2011). *Análisis y Diseño de Sistemas de información*. México: Pearson.
- Pressman, R. (2010). *Ingeniería del software, un enfoque práctico* (Séptima edición). Bogotá: McGraw-Hill.
- P. Bourque y R.E. Fairley, (eds) (2014)., *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society.
- Rodríguez Mogel, E. (2005). *Metodología de la investigación*. Villahermosa - México: Universidad de Juarez, Autónoma de Tabasco.

7. CONTROL DEL DOCUMENTO

	Nombre			Cargo	Dependencia	Fecha
Autor (es)	Carlos	Enrique	Navia	Instructor	CCyT	Julio de 2024
	Torres					

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Estructura condicional

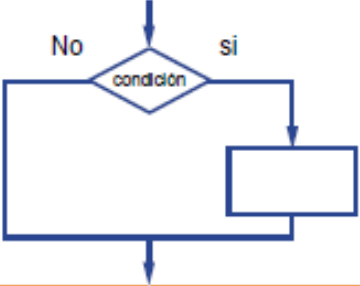
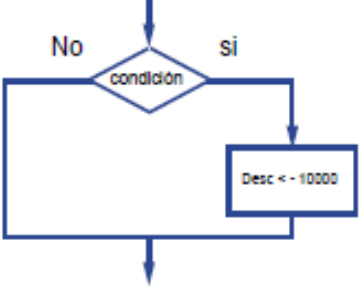
Condicionales IF

Los condicionales tienen la siguiente estructura. Ten en cuenta que lo que contiene los paréntesis es la comparación que debe cumplir para que los elementos se cumplan.

```
if ( a > b ):  
    elementos  
elif ( a == b ):  
    elementos  
else:  
    elementos
```

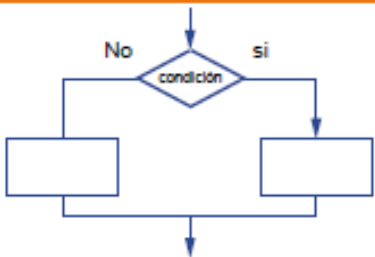
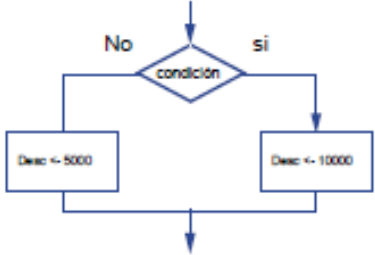
Estructura condicional - Simple

Sintaxis de una estructura condicional simple

Diagrama de Flujo	LPP
	<pre>Si condición Entonces xxxxxxx Fin Si</pre>
Ejemplo Diagrama de Flujo	Ejemplo LPP
	<pre>Si edad < 18 Entonces Desc < - 10000 Fin Si</pre>

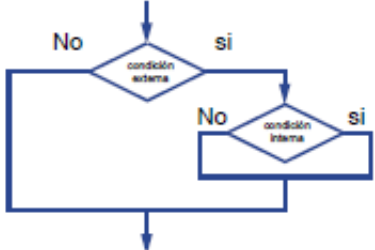
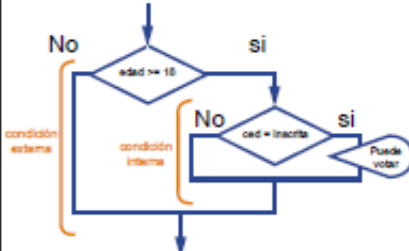
Estructura condicional - Compuesta

Sintaxis de una estructura condicional compuesta

Diagrama de Flujo	LPP
	<pre>Si condición Entonces xxxxxxx Sino yyyyyy Fin Si</pre>
Ejemplo Diagrama de Flujo	Ejemplo LPP
	<pre>Si edad < 18 Entonces Desc <- 10000 Sino Desc <- 5000 Fin Si</pre>

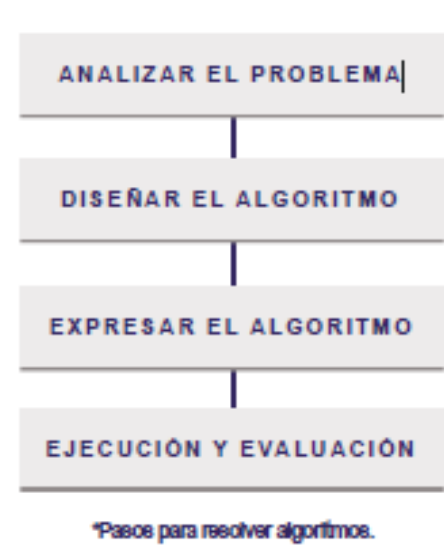
Estructura condicional - Compuesta

Sintaxis de una estructura condicional anidada

Diagrama de Flujo	LPP
	<pre>Si condiciónExt Entonces Si condiciónInt Entonces XXXXXXXX Fin Si Sino yyyyyyy Fin Si</pre>
Ejemplo Diagrama de Flujo	Ejemplo LPP
	<pre>Si edad >= 18 Entonces Si ced = inscrita Escriba "Puede Votar" Fin Si Fin Si</pre>

Resolución de problemas con algoritmos

- Para solucionar un problema mediante un algoritmo es necesario seguir un orden, uno de los primeros pasos es el diseño previo de un algoritmo, con la resolución de problemas se puede seguir este orden de la siguiente manera:



Pasos para Analizar un problema.

- **Reformular el problema:** si el problema que se ha planteado no se encuentra bien formulado, redactado o no se entiende al momento de leerse, debe reformularlo de tal manera que usted lo entienda y tenga claro la situación problema a solucionar.



Pasos para Analizar un problema.

- **Resultados Esperados:** debe especificar, describir y/o escribir los resultados que espera, por ejemplo (cuál es el producto final que se quiere tener para dar solución al problema, cual es la información a la que se necesita llegar, que se espera del problema citado).



Pasos para Analizar un problema.

- **Restricciones:** determina las condiciones que plantea el problema para lograr el resultado, lo que está permitido, lo prohibido.
- **Procesos necesarios:** en esta fase debe definir los procesos para poder convertir la información disponible, en resultados esperados que den solución al problema ya que se determinan los procesos que se necesitan, las formulas a utilizar y el orden de lo que se debe realizar.



Pasos para Analizar un problema.

- **Datos disponibles:** identificar la información disponible se resuelve haciéndose las siguientes preguntas: ¿qué información es importante o relevante para solucionar el problema? ¿cuáles son los datos de entrada?, ¿cuál es la incógnita?, ¿qué información me falta para resolver el problema?.



Ejemplo Explicativo.

- José David, es un muchacho que desea comprar una maleta de \$105.000. El recibe ingresos de diferentes fuentes: en la casa le dan \$6.000 para sus gastos semanales durante 4 semanas, por atender una tienda, tres veces recibió \$12.000. También su hermano lava la piscina una vez al mes por \$13.000 y cuida la tienda por \$10.000. ¿José David tiene ahorrado el dinero suficiente para comprar la maleta o aún le falta?
- Aplique los pasos explicados anteriormente para el análisis del problema.

Ejemplo.

Formular el problema:	
Resultados esperados:	Saber salario mensual, numero de horas extra, valor de horas extras, salario total
Datos disponibles:	Ingreso horas trabajadas, valor por horas,
Restricciones:	las horas extras se calculan a 1.5 veces la hora normal de trabajo. Si el trabajador se pasa de 160 h/m calcular el numero de h/ex
Procesos:	Si horas trabajadas es mayor a 160 <ul style="list-style-type: none">• numero de horas extra=números de horas trabajadas-160• valor h/ex=numero de horas extras * (valor horas * 1.5)• salario mensual=160*valor por horas• salario total=salario mensual+ valor h/ex si no <ul style="list-style-type: none">• salario mensual=160*valor por horas

Ejemplo.

Nombre de la variable	Descripción
salarioMen	Salario mensual
numHrasExt	Numeros de horas extras
valHrasExt	Valor horas extras
salTotal	Salario total
hrasTrab	Horas trabajadas

Ejercicios de algoritmos condicionales.

- Sistema de salud del jugador: Implementa un sistema que muestre un mensaje diferente según el nivel de salud del jugador (100%, 75%, 50%, 25%, 0%).

▼ Solucion.

Análisis del Problema

Aspecto	Descripción
Reformular el problema	Implementar un sistema que evalúe la salud de un jugador y muestre un mensaje diferente cuando la salud se encuentre en los niveles del 100%, 75%, 50%, 25%, y 0%.
Resultados esperados	Se espera que, dependiendo del nivel de salud del jugador, se muestre un mensaje adecuado. El sistema deberá evaluar la salud y devolver el mensaje correspondiente.
Datos disponibles	La salud del jugador varía entre 0 y 100. Los niveles importantes son 100%, 75%, 50%, 25%, y 0%. La incógnita es qué mensaje mostrar en función del valor actual de la salud.
Restricciones	El sistema solo evaluará los valores 100, 75, 50, 25, y 0. Cualquier otro valor será ignorado o tratado como un valor intermedio que no genera un mensaje.
Procesos	Se deben usar condiciones para comparar el nivel de salud del jugador con los valores específicos (100, 75, 50, 25, 0). En función de la comparación, se determinará el mensaje a mostrar.

Definición de Variables

Nombre de la variable	Descripción
saludJugador	Representa el nivel de salud del jugador, que varía entre 0 y 100.
mensajeSalud	Almacena el mensaje correspondiente según el nivel de salud actual del jugador.

Solución del Algoritmo por cada paso del proceso.

Paso	Parte	Descripción
1	Entrada	Se obtiene el nivel de salud del jugador (saludJugador).
2	Proceso	Evaluar el nivel de salud usando condicionales. Si saludJugador es 100, 75, 50, 25 o 0, asignar un mensaje específico a la variable mensajeSalud .
3	Salida	Mostrar el mensaje almacenado en mensajeSalud al jugador.

- Selección de dificultad: Crea un menú que permita al jugador elegir entre tres niveles de dificultad (fácil, normal, difícil) y muestre un mensaje correspondiente.
- Cambio de arma: Diseña un sistema que permita al jugador cambiar entre 4 armas diferentes y muestre las características del arma seleccionada.
- Sistema de daño: Implementa un cálculo de daño basado en el tipo de ataque (ligero, medio, pesado) y la armadura del enemigo.
- Interacción con objetos: Crea un sistema que permita al jugador interactuar con diferentes objetos (puerta, cofre, palanca) y muestre el resultado de la interacción.
- Diálogo simple: Implementa un sistema de diálogo donde el jugador puede elegir entre 3 respuestas y el NPC reaccione de manera diferente a cada una.
- Sistema de stamina: Diseña un sistema que muestre diferentes mensajes según el nivel de stamina del jugador (100%, 75%, 50%, 25%, 0%).
- Efectos de estado: Crea un sistema que aplique diferentes efectos al jugador (envenenado, quemado, congelado, normal) y muestre el estado actual.
- Cambio de clima: Implementa un sistema que cambie el clima del juego (soleado, lluvioso, nevado, tormentoso) y muestre un mensaje descriptivo.
- Sistema de experiencia: Diseña un sistema que muestre el nivel actual del jugador basado en su experiencia (novato, aprendiz, experto, maestro).
- Compra de objetos: Crea un sistema simple de tienda donde el jugador pueda comprar hasta 3 objetos diferentes según su oro disponible.
- Habilidades especiales: Implementa un sistema que permita al jugador usar una habilidad especial según su clase (guerrero, mago, arquero) y muestre un mensaje con las características de la habilidad.
- Modificadores de ataque: Crea un sistema que aplique diferentes modificadores al ataque del jugador según las condiciones del entorno (día, noche, lluvia, niebla) muestre un mensaje que explique las condiciones.
- Sistema de afinidad: Implementa un sistema que muestre diferentes reacciones de un NPC según el nivel de afinidad con el jugador (hostil, neutral, amigable).
- Efectos de pociones: Diseña un sistema que aplique diferentes efectos al jugador según la poción consumida (salud, maná, fuerza, velocidad).
- Trampas: Crea un sistema que determine el tipo de trampa activada (pinchos, dardos, foso) y el daño causado al jugador.
- Desbloqueo de habilidades: Implementa un sistema que desbloquee una nueva habilidad para el jugador según su nivel actual.
- Cambio de apariencia: Diseña un sistema que cambie la descripción de la apariencia del personaje según el equipamiento seleccionado (armadura ligera, media, pesada).

- Reacción de enemigos: Crea un sistema que determine la reacción de un enemigo (atacar, huir, llamar refuerzos) según la diferencia de nivel con el jugador, muestra un mensaje que explique la reacción.

Template con ejemplo de análisis de algoritmos

Proceso de análisis y desarrollo de algoritmos.



Los aprendices de forma individual desarrollan los ejercicios planteados según el proceso de análisis de algoritmos, una vez analizados realizar diagramas de flujo usando la herramienta: <https://app.diagrams.net/?mode=google>

Informe de cierre entrenamiento.

El equipo esta conformado por:

NOMBRE	PRIMER APELLIDO	SEGUNDO APELLIDO	PROGRAMA DE FORMACION
JUAN GABRIEL	ERASO	ARIAS	TGO DESARROLLO DE VIDEOJUEGOS FICHA: 2617573
EDWIN	VIÑA	RAMOS	TGO DESARROLLO DE VIDEOJUEGOS FICHA: 2617573

Instructor entrenador:

Carlos Enrique Navia Torres.

Casos de Estudio entrenamiento asignado.

[Brief para Coffee Tasting VR Experience-final.pdf](#)

Evidencias cierre entrenamiento.

Los aprendices candidatos a la competencia, cierran con su proceso de entrenamiento, desarrollando los siguientes procesos:

Video Gameplay final prototipo del proyecto.

El martes 17 de septiembre se realizó la entrega del video que muestra el gameplay del proyecto. Dicho video fue evaluado y, aunque se hicieron algunas sugerencias de mejora, en términos generales se evidenció el cumplimiento de los objetivos definidos para este proceso de entrenamiento.

https://drive.google.com/file/d/1TgrTWSVTLfNrC24YBKgzkBVJEnN1n9Lm/view?usp=drive_link

Por otro lado, se llevó a cabo un ejercicio de presentación de pitch, por el aprendiz Edwin Viña, en el cual se hicieron algunas recomendaciones para su mejora, por que en términos generales, cumplió con los requisitos y objetivos establecidos en el proceso de entrenamiento.

https://youtu.be/reTdzvpEoHU?si=RcXbqKVkOhaCbb_V

Finalmente, quiero destacar la entrega y dedicación que los aprendices han demostrado a lo largo de su proceso de entrenamiento, lo cual ha sido fundamental para alcanzar los resultados obtenidos.

Taller de conceptualización manejo de colecciones en Java.

- *Colecciones de Datos*
- Que son los *List en las colecciones de datos en Java*
- Que metodos manejo la clase List
- Que son los *ArrayList*
- Como instanciar un *ArrayList*.
- Implementar ejemplos de uso con el proyecto SENA viewer



Los aprendices de forma individual desarrollan la revision de conceptos planteados, crear un documento que debe contener normas APA 7th y la siguiente estructura:

- Presentación (Portada)
- Introducción
- Desarrollo del trabajo
- Conclusiones
- Genera la implementación de colección de datos en el proyecto y genera un commit en con las implementaciones.

Segundo Informe avance entrenamiento.

El equipo esta conformado por:

NOMBRE	PRIMER APELLIDO	SEGUNDO APELLIDO	PROGRAMA DE FORMACION
JUAN GABRIEL	ERASO	ARIAS	TGO DESARROLLO DE VIDEOJUEGOS FICHA: 2617573
EDWIN	VIÑA	RAMOS	TGO DESARROLLO DE VIDEOJUEGOS FICHA: 2617573

Instructor entrenador:

Carlos Enrique Navia Torres.

Casos de Estudio entrenamiento asignado.

[Brief para Coffee Tasting VR Experience-final.pdf](#)

Evidencias Avances hasta el momento.

Los aprendices candidatos a la competencia, continuan con su proceso de entrenamiento, desarrollando los siguientes procesos:

Proceso de pruebas de prototipo del proyecto.

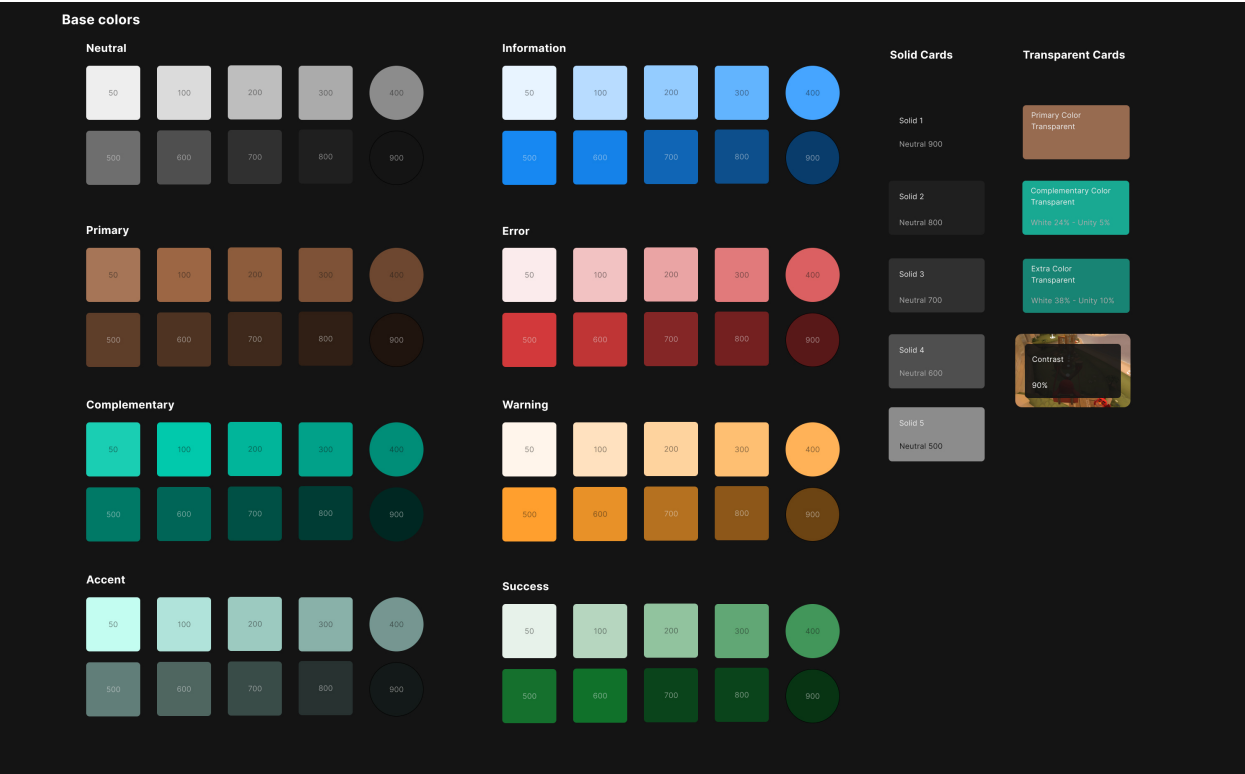
El jueves 5 de septiembre se realizaron pruebas de avance del proyecto, en las cuales el líder del área, junto con algunos instructores, evaluaron el prototipo del proyecto interactivo propuesto.

https://drive.google.com/file/d/1BefFkcNdUJqzsTMjTrKj4cbON1cwKiIK/view?usp=drive_link

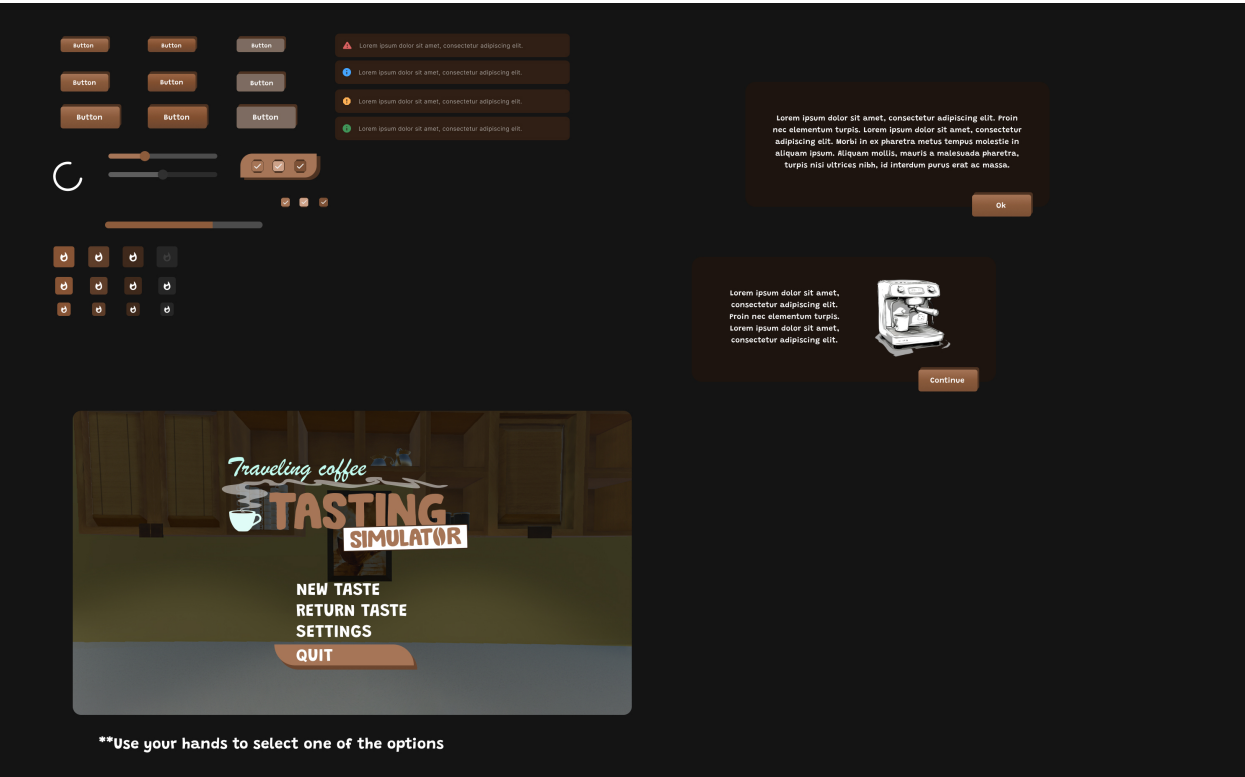
https://drive.google.com/file/d/1_ReHHWLs8NswpHaRD4HibOOL6NEhcxZu/view?usp=drive_link

Por otro lado, se consolidó el diseño del sistema de diseño y los componentes de las interfaces

Desing System



Ui Kit components



Game play implementado en el proyecto.

<https://youtu.be/qVym8xOsAk8?si=wrnYB3BqLpw8CvjS>

Relación entre clases en Java.

1. Definición de Clases y Relaciones

- **Factura** : Representa la factura generada por la compra de productos.
- **Cliente** : Representa al cliente que realiza la compra.
- **ÍtemFactura** : Representa cada línea de la factura, es decir, el detalle de un producto específico y su cantidad.
- **Producto** : Representa el producto que el cliente compra.

2. Diagrama de Clases Simplificado

- **Cliente** tiene una relación *uno a muchos* con **Factura** .
- **Factura** tiene una relación *uno a muchos* con **ÍtemFactura** .
- **ÍtemFactura** tiene una relación *uno a uno* con **Producto** .

▼ 3. Definición de Clases y atributos en código Java

▼ Clase Cliente

```
public class Cliente {  
    private String nombre;  
    private String nif; // Número de Identificación Fiscal  
}
```

▼ Clase Producto

```
public class Producto {  
    private String codigo;  
    private String nombre;  
    private double precio;  
}
```

▼ Clase ÍtemFactura

```
public class ÍtemFactura  
    private Producto producto;  
    private int cantidad;  
}
```

▼ Clase Factura

```
import java.util.Date;  
  
public class Factura {  
    private int folio;
```

```
private String descripcion;  
private Date fecha;  
private Cliente cliente;  
private ItemFactura[] items;  
}
```

▼ 4. Crear Getter y Setters de las clases.

▼ Clase Cliente

```
public class Cliente {  
    private String nombre;  
    private String nif; // Número de Identificación Fiscal  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getNif() {  
        return nif;  
    }  
  
    public void setNif(String nif) {  
        this.nif = nif;  
    }  
}
```

▼ Clase Producto

```
public class Producto {  
    private String codigo;  
    private String nombre;  
    private double precio;  
  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {
```

```

        this.nombre = nombre;
    }

    public float getPrecio() {
        return precio;
    }

    public void setPrecio(float precio) {
        this.precio = precio;
    }
}

```

▼ Clase ÍtemFactura

```

public class ÍtemFactura
    private Producto producto;
    private int cantidad;

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }

    public Producto getProducto() {
        return producto;
    }

    public void setProducto(Producto producto) {
        this.producto = producto;
    }
}

```

▼ Clase Factura

```

import java.util.Date;

public class Factura {
    private int folio;
    private String descripcion;
    private Date fecha;
    private Cliente cliente;
    private ItemFactura[] items;

    public int getFolio() {
        return folio;
    }
}

```

```

    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public ItemFactura[] getItems() {
        return items;
    }

}

```

▼ 5. Crear constructores.

▼ Clase Cliente

```

public class Cliente {
    private String nombre;
    private String nif; // Número de Identificación Fiscal

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

```

    public String getNif() {
        return nif;
    }

    public void setNif(String nif) {
        this.nif = nif;
    }
}

```

▼ Clase Producto

```

public class Producto {
    private String codigo;
    private String nombre;
    private double precio;
    /**
     * El último código asignado a un producto.
     * Este campo se usa para generar nuevos códigos únicos para cada instancia de {@code Producto}.
     */
    private static int ultimoCodigo;

    /**
     * Crea un constructor que asigna automáticamente un código único al producto incrementando {@code ultimoCodigo}.
     */
    public Producto() {
        this.codigo = ++ultimoCodigo;
    }

    public int getCodigo() {
        return codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public float getPrecio() {
        return precio;
    }

    public void setPrecio(float precio) {

```

```

        this.precio = precio;
    }

    /**
     * Devuelve una representación en cadena del producto.
     *
     * <p>
     * La representación incluye el código, nombre y precio del producto, separados por tabulaciones.
     * </p>
     *
     * @return Una cadena que representa el producto.
     */
    @Override
    public String toString() {
        return codigo +
            "\t" + nombre +
            "\t" + precio;
    }
}

```

▼ Clase ÍtemFactura

```

public class ÍtemFactura
{
    private Producto producto;
    private int cantidad;

    // crea constructor para clase ItemFactura
    public ÍtemFactura(int cantidad, Producto producto)
    {
        this.cantidad = cantidad;
        this.producto = producto;
    }

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }

    public Producto getProducto() {
        return producto;
    }

    public void setProducto(Producto producto) {
        this.producto = producto;
    }
}

```

```

    }
}

```

▼ Clase Factura

```

import java.util.Date;

public class Factura {
    private int folio;
    private String descripcion;
    private Date fecha;
    private Cliente cliente;
    private ItemFactura[] items;

    /**
     * Índice actual en el arreglo de ítems.
     */
    private int indiceItems;

    /**
     * Número máximo de ítems que una factura puede contener.
     */
    public static final int MAX_ITEMS = 12;

    /**
     * El último folio asignado a una factura.
     * Este campo se usa para generar nuevos folios únicos para cada instancia de {@code Factura}.
     */
    private static int ultimoFolio;

    /**
     * Crea una nueva instancia de {@code Factura}.
     * <p>
     * El constructor inicializa la descripción, el cliente, y el arreglo de ítems. Asigna un folio único a la factura e
     * inicializa la fecha de emisión a la fecha actual.
     * </p>
     *
     * @param descripcion La descripción de la factura.
     * @param cliente El cliente asociado a la factura.
     */
    public Factura(String descripcion, Cliente cliente) {

        this.descripcion = descripcion;
        this.cliente = cliente;
        this.items = new ItemFactura[MAX_ITEMS];
    }
}

```

```

        this.folio = ++ultimoFolio;
        this.fecha = new Date();
    }

    public int getFolio() {
        return folio;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public ItemFactura[] getItems() {
        return items;
    }

}

```

▼ 5. Crear metodos y desarrollar logica para ejecutar la aplicación

▼ Clase ÍtemFactura

```

public class ÍtemFactura
    private Producto producto;
    private int cantidad;

    // crea constructor para clase ItemFactura
    public ItemFactura(int cantidad, Producto producto)

```

```

{
    this.cantidad = cantidad;
    this.producto = producto;
}

public int getCantidad() {
    return cantidad;
}

public void setCantidad(int cantidad) {
    this.cantidad = cantidad;
}

public Producto getProducto() {
    return producto;
}

public void setProducto(Producto producto) {
    this.producto = producto;
}

// metodo para calcular el importe de un producto.
public float calcularImporte(){
    return this.cantidad * this.producto.getPrecio
();
}

// metodo para mostrar informacion del item registra
do.
@Override
public String toString() {
    return producto +
        "\t" + cantidad +
        "\t" + calcularImporte();
}
}

```

▼ Clase Factura

```

import java.util.Date;

public class Factura {
    private int folio;
    private String descripcion;
    private Date fecha;
    private Cliente cliente;
    private ItemFactura[] items;

    /**

```

```

        * Índice actual en el arreglo de ítems.
        */
        private int indiceItems;
        /**
         * Número máximo de ítems que una factura pu
         ede contener.
         */
        public static final int MAX_ITEMS = 12;

        /**
         * El último folio asignado a una factura.
         * Este campo se usa para generar nuevos fol
         ios únicos para cada instancia de {@code Factura}.
         */
        private static int ultimoFolio;

    /**
     * Crea una nueva instancia de {@code Factura}.
     * <p>
     * El constructor inicializa la descripción, el clie
     nte, y el arreglo de ítems. Asigna un folio único a la f
     actura e
     * inicializa la fecha de emisión a la fecha actual.
     * </p>
     *
     * @param descripcion La descripción de la factura.
     * @param cliente El cliente asociado a la factura.
     */
    public Factura(String descripcion, Cliente clien
    te) {

        this.descripcion = descripcion;
        this.cliente = cliente;
        this.items = new ItemFactura[MAX_ITEMS];
        this.folio = ++ultimoFolio;
        this.fecha = new Date();
    }

    public int getFolio() {
        return folio;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

```

```

    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public ItemFactura[] getItems() {
        return items;
    }

    // añade los items de la factura
    public void addItemFactura(ItemFactura item) {
        if (indiceItems < MAX_ITEMS) {
            this.items[indiceItems++] = item;
        }
    }

    // calcula el total de la factura
    public float calcularTotal() {
        float total = 0.0f;
        for (ItemFactura item : this.items) {
            if (item == null) {
                continue;
            }
            total += item.calcularImporte();
        }
        return total;
    }

    // prepara el detalle de la factura
    public String generarDetalle() {
        StringBuilder sb = new StringBuilder("Factura N
º: ");
        sb.append(folio)
            .append("\nCliente: ")
            .append(this.cliente.getNombre())
            .append("\t NIF: ")

```

```

        .append(cliente.getNif())
        .append("\nDescripción: ")
        .append(this.descripcion)
        .append("\n");

        SimpleDateFormat df = new SimpleDateFormat("dd
'de' MMMM, yyyy");
        sb.append("Fecha Emisión: ")
            .append(df.format(this.fecha))
            .append("\n")
            .append("\n#\tNombre\t$\tCant.\tTotal
\n");

        for(ItemFactura item: this.items){
            if(item == null){
                continue;
            }
            sb.append(item)
                .append("\n");
        }
        sb.append("\nGran Total: ")
            .append(calcularTotal());

        return sb.toString();
    }

    @Override
    public String toString() {
        return generarDetalle();
    }
}

```

▼ Clase Main

```

import com.cenavia.appfacturas.model.*;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Cliente cliente = new Cliente();
        cliente.setNif("5555-5");
        cliente.setNombre("Andrés");
    }
}

```

```

Scanner s = new Scanner(System.in);
System.out.print("Ingrese una descripción de la fa
String desc = s.nextLine();
Factura factura = new Factura(desc, cliente);

Producto producto;

System.out.println();

for(int i = 0; i<5; i++){
    producto = new Producto();
    System.out.print("Ingrese producto nº " + produ
    producto.setNombre(s.nextLine());

    System.out.print("Ingrese el precio: ");
    producto.setPrecio(s.nextFloat());

    System.out.print("Ingrese la cantidad: ");
    factura.addItemFactura(new ItemFactura(s.nextI

    System.out.println();
    s.nextLine();
}
System.out.println(factura);
}
}

```

Ejercicios Operadores logicos-solve.

Ejercicio 1:

Descripción: Estás desarrollando un juego de plataformas en el que un personaje debe saltar obstáculos. Si el personaje tiene más de 3 vidas y ha recolectado al menos 5 monedas, puede realizar un salto doble.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si el personaje puede realizar el salto doble?

Respuesta esperada: `(vidas > 3) && (monedas >= 5)`

Ejercicio 2:

Descripción: En un juego de estrategia, si el nivel de recursos de un jugador es menor que 100, se activará un evento de escasez.

Pregunta: ¿Qué expresión lógica permite activar el evento de escasez?

Respuesta esperada: `recursos < 100`

Ejercicio 3:

Descripción: En un juego de carreras, se activa un turbo si la velocidad actual del vehículo es igual o superior a 200 km/h, y si la duración del turbo es menor o igual a 5 segundos.

Pregunta: ¿Qué expresión lógica permite activar el turbo?

Respuesta esperada: `(velocidad >= 200) && (duracionTurbo <= 5)`

Ejercicio 4:

Descripción: En un juego de combate, un personaje puede usar una habilidad especial si su nivel de energía es mayor a 50, pero solo si su barra de vida no está al máximo.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si el personaje puede usar la habilidad especial?

Respuesta esperada: `(energia > 50) && (vida < vidaMaxima)`

Ejercicio 5:

Descripción: En un juego de simulación de construcción, se permite construir un edificio si el jugador tiene al menos 1000 unidades de madera o 500 unidades de piedra.

Pregunta: ¿Qué expresión lógica permite construir el edificio?

Respuesta esperada: `(madera >= 1000) || (piedra >= 500)`

Ejercicio 6:

Descripción: En un juego de rol, un personaje puede atacar a un enemigo si su nivel de ataque es mayor o igual que el nivel de defensa del enemigo.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si el personaje puede atacar al enemigo?

Respuesta esperada: `ataque >= defensaEnemigo`

Ejercicio 7:

Descripción: En un juego de aventuras, se puede abrir una puerta secreta si el jugador tiene una llave de oro y la dificultad del nivel es menor que 7.

Pregunta: ¿Qué expresión lógica permite abrir la puerta secreta?

Respuesta esperada: `(tieneLlaveOro == true) && (dificultad < 7)`

Ejercicio 8:

Descripción: En un juego de simulación, un cultivo está listo para cosechar si ha pasado el tiempo de maduración y la calidad del suelo es mayor o igual a 75.

Pregunta: ¿Qué expresión lógica indica que el cultivo está listo para cosechar?

Respuesta esperada: `(tiempo >= tiempoMaduracion) && (calidadSuelo >= 75)`

Ejercicio 9:

Descripción: En un juego de disparos, un jugador puede recargar su arma si el cargador está vacío o si quedan menos de 3 balas en el cargador.

Pregunta: ¿Qué expresión lógica permite recargar el arma?

Respuesta esperada: `(cargador == 0) || (balas < 3)`

Ejercicio 10:

Descripción: En un juego de supervivencia, el jugador debe construir un refugio si la temperatura ambiente es menor o igual a 5 grados Celsius y está lloviendo.

Pregunta: ¿Qué expresión lógica determina si el jugador debe construir un refugio?

Respuesta esperada: `(temperatura <= 5) && (lloviendo == true)`

Ejercicio 11:

Descripción: En un juego de lucha, un personaje puede realizar un ataque crítico si su velocidad es mayor que la del enemigo y si su fuerza es mayor o igual a 80.

Pregunta: ¿Qué expresión lógica permite realizar el ataque crítico?

Respuesta esperada: `(velocidad > velocidadEnemigo) && (fuerza >= 80)`

Ejercicio 12:

Descripción: En un juego de rol, el jugador puede equipar una armadura especial si su nivel es mayor o igual a 10, pero solo si no está usando otra armadura.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si el jugador puede equipar la armadura especial?

Respuesta esperada: `(nivel >= 10) && (usandoArmadura == false)`

Ejercicio 13:

Descripción: En un juego de estrategia en tiempo real, una unidad puede realizar un ataque a distancia si la distancia al objetivo es menor que 15 unidades y su puntería es mayor o igual al 70%.

Pregunta: ¿Qué expresión lógica permite realizar el ataque a distancia?

Respuesta esperada: `(distanciaObjetivo < 15) && (punteria >= 70)`

Ejercicio 14:

Descripción: En un juego de exploración, una trampa se activa si el jugador tiene menos de 3 puntos de salud y está en una zona peligrosa.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si la trampa se activa?

Respuesta esperada: `(salud < 3) && (zonaPeligrosa == true)`

Ejercicio 15:

Descripción: En un juego de carreras, un vehículo es descalificado si el tiempo total de la carrera es mayor a 10 minutos o si ha recibido más de 3 penalizaciones.

Pregunta: ¿Qué expresión lógica determina si el vehículo es descalificado?

Respuesta esperada: `(tiempoTotal > 10) || (penalizaciones > 3)`

Estos ejercicios permiten a los estudiantes practicar el uso de operadores relacionales en situaciones reales de desarrollo de videojuegos.

Ejercicio 16:

Descripción: En un juego de exploración espacial, la nave puede activar su escudo de protección si su nivel de energía es mayor que 50% y su velocidad es menor que 20 unidades.

Pregunta: ¿Qué expresión lógica se utiliza para activar el escudo de protección?

Respuesta esperada: `(energia > 50) && (velocidad < 20)`

Ejercicio 17:

Descripción: En un juego de defensa de torres, una torre puede atacar a un enemigo si la distancia al enemigo es menor o igual a su rango de ataque y si el enemigo tiene más de 0 puntos de vida.

Pregunta: ¿Qué expresión lógica se utiliza para que la torre ataque?

Respuesta esperada: `(distanciaEnemigo <= rangoAtaque) && (vidaEnemigo > 0)`

Ejercicio 18:

Descripción: En un juego de aventuras, un cofre del tesoro puede abrirse si el jugador tiene la llave correcta y si ha derrotado al guardián del tesoro.

Pregunta: ¿Qué expresión lógica permite abrir el cofre del tesoro?

Respuesta esperada: `(tieneLlave == true) && (guardianDerrotado == true)`

Ejercicio 19:

Descripción: En un juego de simulación de vida, un personaje puede cocinar si su nivel de hambre es mayor o igual a 30 y si tiene al menos 1 ingrediente en su inventario.

Pregunta: ¿Qué expresión lógica permite que el personaje cocine?

Respuesta esperada: `(hambre >= 30) && (ingredientes > 0)`

Ejercicio 20:

Descripción: En un juego de rol multijugador, un jugador puede unirse a una partida si su nivel es mayor o igual a 15 y si hay menos de 10 jugadores en la partida actual.

Pregunta: ¿Qué expresión lógica se utiliza para determinar si el jugador puede unirse a la partida?

Respuesta esperada: `(nivel >= 15) && (jugadoresEnPartida < 10)`

Ejercicio 21:

Descripción: En un juego de acción, una trampa se desactiva si el personaje tiene un objeto específico en su inventario o si ha completado la misión asociada a la trampa.

Pregunta: ¿Qué expresión lógica permite desactivar la trampa?

Respuesta esperada: `(tieneObjeto == true) || (misionCompletada == true)`

Ejercicio 22:

Descripción: En un juego de puzzles, una puerta se abre si el jugador resuelve al menos 3 acertijos correctamente y si su tiempo restante es mayor que 5 minutos.

Pregunta: ¿Qué expresión lógica permite abrir la puerta?

Respuesta esperada: `(acertijosResueltos >= 3) && (tiempoRestante > 5)`

Ejercicio 23:

Descripción: En un juego de simulación de economía, un jugador puede comprar un ítem si tiene más de 500 monedas o si tiene un bono de descuento válido.

Pregunta: ¿Qué expresión lógica permite realizar la compra?

Respuesta esperada: `(monedas > 500) || (bonoDescuentoValido == true)`

Ejercicio 24:

Descripción: En un juego de carreras, un vehículo puede activar un salto si su velocidad es mayor que 150 km/h y si está en una rampa.

Pregunta: ¿Qué expresión lógica permite activar el salto?

Respuesta esperada: `(velocidad > 150) && (enRampa == true)`

Ejercicio 25:

Descripción: En un juego de supervivencia, un personaje puede encender una fogata si tiene al menos 1 unidad de madera y si la temperatura es menor o igual a 0 grados Celsius.

Pregunta: ¿Qué expresión lógica permite encender la fogata?

Respuesta esperada: `(madera >= 1) && (temperatura <= 0)`

Estos ejercicios adicionales ayudan a explorar más situaciones comunes en el desarrollo de videojuegos donde los operadores relacionales juegan un papel crucial en la toma de decisiones lógicas.

Taller conceptual sobre AWT yJava Swing (GUI).

Descripción:

Que los Aprendices comprendan los conceptos fundamentales del desarrollo de interfaces gráficas en Java utilizando AWT y Swing, y sean capaces de identificar y aplicar los componentes y patrones básicos de diseño para la creación de aplicaciones gráficas.

1. Introducción a Java y GUI

- Historia de Java y su ecosistema.
- Diferencias entre AWT y Swing.
- Ventajas de usar Swing sobre AWT.

2. Introducción a AWT (Abstract Window Toolkit)

Define e indica cómo se usan:

- Contenedores y componentes.
- Uso de `Frame`, `Panel`, `Button`, `Label`, `TextField`.
- Manejo de layout managers: `FlowLayout`, `BorderLayout`, `GridLayout`.
- Gestión de eventos: modelo de delegación de eventos en AWT.
- Implementación de interfaces de eventos (`ActionListener`, `MouseListener`, etc.).

3. Introducción a Swing

Define e indica cómo se usa:

- La estructura básica de una aplicación Swing.
- Diferencias entre contenedores de nivel superior: `JFrame`, `JDialog`, `JApplet`.
- Componentes básicos: `JButton`, `JLabel`, `TextField`, `JTextArea`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JList`.
- Uso de `JPanel` y diseño modular de interfaces.
- Introducción a `JMenu` y barras de menú (`JMenuBar`, `JMenuItem`).

4. Layouts en Swing

Define e indica cómo se usa:

- Los layout managers: `BoxLayout`, `GridBagLayout`, `GroupLayout`.
- Diseño de interfaces complejas utilizando paneles anidados.

5. Gestión de Eventos en Swing

Define e indica cómo se usa:

- Concepto de listeners y adaptadores.
- Eventos comunes en Swing: `ActionEvent` , `MouseEvent` , `KeyEvent` .

6. Componentes Avanzados de Swing

Define e indica cómo se usa:

- `JTable` para representar datos tabulares.
- Creación y manejo de `JTree` para datos jerárquicos.
- Uso de `JTabbedPane` para interfaces con pestañas.
- Implementación de diálogos modales con `JOptionPane` .
- Barras de progreso (`JProgressBar`) y sliders (`JSlider`).



Los aprendices de forma individual desarrollan la revisión de conceptos planteados, crear un documento que debe contener normas APA 7th y la siguiente estructura:

- Presentación (Portada)
- Introducción
- Desarrollo del trabajo - se debe mostrar ejemplo de código en los casos requeridos.
- Conclusiones