PROCESO GESTIÓN DEL TALENTO HUMANO

FORMATO INFORME MENSUAL EJECUCIÓN CONTRACTUAL

Barranquilla, Junio de 2020

Señor (a)
MANUEL GREGORIO HORMECHEA LANCE
SUPERVISOR(A) CONTRATO No. **1333805**
Coordinador de Automatización
Automatización
Barranquilla

Asunto: Informe mensual de ejecución contractual Mes Junio del año 2020

Referencia: No 1333805 del año 2020

RICARDO ENRIQUE CANTILLO CARRILLO, identificado con la cédula de ciudadanía No. 72.201.581 de Barranquilla, en mi calidad de Contratista del SENA, en Automatización, en cumplimiento del Contrato de Prestación de Servicios de la referencia, a continuación presento el Informe de actividades realizadas en el mes objeto de cobro.

Valor y forma de Pago: e). un pago correspondiente al mes de Junio de 2020 por valor de (3.278.181). Los honorarios serán pagados por el SENA al contratista de acuerdo al cronograma definido por la Dirección Administrativa y Financiera de la Dirección General, en la cuenta de Ahorros No. 0098000200063852 BBVA, cuyo titular es el Contratista.

Plazo: Será hasta el 30 de Junio de 2020.

OBJETO:
Prestación de servicios profesionales como instructor para impartir formación profesional titulada y complementaria en el área de teleinformática - análisis y desarrollo de sistemas de información de acuerdo a la programación académica del SENA Centro Nacional Colombo Alemán

CLÁUSULA (No. de cláusula donde se encuentren las obligaciones del contratista)

No	Obligaciones	Acciones realizadas	Evidencias
1	Programar las actividades de enseñanza, aprendizaje y evaluación de conformidad con los de formación y el calendario institucional y el Manual de procedimientos para la ejecución.	REVISIÓN DE ACTIVIDADES, INTERACCIÓN VIRTUAL CON APRENDICES DE LA FORMACIÓN, ENVÍO DE CORREOS DE FOROS DE BUZÓN DE DUDAS E INQUIETUDES. LABORATORIO DE PROGRAMACIÓN, DEL	NODO TIC, de los aprendices de los programas de ADSI, realización de los Laboratorios de Rest y Esquemas de SQL. Programación con

		NODO TIC DEL CNCA - PISO 6. SALA ADSI 2 DE PROGRAMACIÓN.	frameWorks SpringBoot

A continuación relaciono los desplazamientos que realicé previo a la presentación de este informe. Una vez finalizado cada desplazamiento presenté al ordenador del gasto el informe en el Formato Informe Legalización Desplazamiento Contratista GTH-F-087, en el que se describieron las actividades desarrolladas y los resultados de cada desplazamiento. Cada informe cuenta con el visto bueno del Supervisor.
Se lista a continuación el soporte de la legalización de los desplazamientos realizados, los cuales forman parte integral del presente informe de ejecución contractual.

ITEM	No DE LA ORDEN DE VIAJE	LUGAR DE DESPLAZAMIENTO	FECHAS DE DESPLAZAMIENTOS	
			INICIO	FINALIZACIÓN

Nota 1: *Por cada desplazamiento que haya realizado el contratista, adjuntará el respectivo informe que la soporte. En caso de haber realizado el desplazamiento en fecha posterior a la presentación del informe de ejecución contractual, deberá reportarlo en el siguiente informe de ejecución contractual.*

Para el trámite de la cuenta me permito adjuntar: Documentos enunciados como evidencias del cumplimiento de las obligaciones contractuales y los desplazamientos realizados y el No. de la planilla, operador y periodo. (Decreto Ley 2106 de 2019 – " *Decreto Ley Antitrámites"*)

Evidencias en ( ) folios

Cordialmente,

RICARDO ENRIQUE CANTILLO CARRILLO
Contratista
C.C. No. 72.201.581

Recibí a satisfacción:

MANUEL GREGORIO HORMECHEA LANCE
Supervisor(a) Contrato 1333805 de 2020
Coordinador de Automatización

SERVICIO NACIONAL DE APRENDIZAJE SENA
INFORME FINAL DE SUPERVISIÓN
CONTRATACIÓN DE PRESTACIÓN DE SERVICIOS PERSONALES

CONTRATO No. 1333805.

NOMBRE DEL CONTRATISTA: RICARDO ENRIQUE CANTILLO CARRILLO

IDENTIFICACION DEL CONTRATISTA: C.C 72.201.581 de Barranquilla

OBJETO DEL CONTRATO: Prestación de servicios profesionales como instructor para impartir formación profesional titulada y complementaria en el área de teleinformática - análisis y desarrollo de sistemas de información de acuerdo a la programación académica del SENA Centro Nacional Colombo Alemán.

OBLIGACIONES DEL CONTRATISTA: 1. Formular proyecto formativo, planeación pedagógica del proyecto y recursos didácticos (Guías de aprendizaje, instrumentos de evaluación y material de apoyo) para el desarrollo de las competencias del programa de formación según lineamientos del procedimiento y guía de desarrollo curricular. 2. Realizar el alistamiento de las actividades de aprendizaje a orientar teniendo en cuenta: ambientes, número de aprendices, disponibilidad de equipos, modalidad de formación, y duraciones establecidas en las guías de aprendizaje para el logro de los resultados de aprendizaje del programa. 3. Realizar las actividades de inducción a los aprendices utilizando el ambiente virtual dispuesto (LMS) acorde a lo establecido en la circular de inducción vigente y guía desarrollo del proceso formativo. 4. Asociar aprendices a la ruta de aprendizaje en el aplicativo Sofia plus dispuesto por la entidad 5. Orientar el desarrollo de las actividades de Aprendizaje promoviendo el aprendizaje significativo, la solución creativa de problemas, el desarrollo de estrategias para el aprendizaje autónomo, el uso de las Tic, el trabajo colaborativo, planteando acciones de refuerzo, mejoramiento y apoyo que motiven y fundamenten al aprendiz para el desarrollo de sus competencias según lo establecido en el procedimiento de ejecución de la formación profesional integral 6. Utilizar herramientas tecnológicas que El SENA brinda para la gestión del aprendizaje y apoyo a los procesos formativos, para organizar y desarrollar actividades de aprendizaje y llevar el registro del portafolio de evidencias. 7. Registrar oportunamente en aquellos casos de inasistencias, deserciones, condicionamientos de matrícula, cancelación de matrícula en el aplicativo SOFIA plus de acuerdo con lo establecido en el Reglamento del Aprendiz vigente. 8. Emitir juicios de evaluación con base en la valoración de las evidencias de Aprendizaje, las características y las premisas de evaluación del aprendizaje descritas en la Guía para desarrollar el proceso formativo, el procedimiento de ejecución de la formación y el reglamento del Aprendiz SENA. 9. Realizar evaluación al desarrollo del programa ante el Equipo Pedagógico del Centro estableciendo pertinencia del programa, índice de deserción, y observaciones para la mejora de los contenidos curriculares del Programa de Formación, dejando este registro mediante Acta. 10. Atender con oportunidad y eficiencia las sugerencias indicadas por el Supervisor del Contrato para el cabal cumplimiento de sus obligaciones contractuales. 11. Apoyar los procesos relacionados con Investigación, Ciencia, Tecnología e Innovación del Centro de Formación en caso de ser requerido o estar interesado en participar 12. Reportar en el sistema SOFIA Plus en un plazo máximo de 3 días, todas las actividades de acuerdo con los procesos que son de su responsabilidad, garantizando la calidad de la información y su coherencia con el proceso formativo, tales como: Registro de los juicios evaluativos, Creación de rutas y asociación de aprendices, Registro de juicios evaluativos del reconocimiento de aprendizajes previos, Cierre de fichas. 13. Cumplir con la entrega de productos a los que se comprometió en la formulación del proyecto aprobado. 14. Participar en las actividades

de divulgación de ciencia y tecnología organizados por los Centros de Formación. 15. Presentar informes mensuales en los cuales se evidencie los avances respecto al cumplimiento de los indicadores, metas u objetivos propuestos para el área y/o proceso en el que interviene; y un informe final al terminar la ejecución del contrato. 16. Utilizar y Cuidar los ambientes, maquinaria, materiales, vehículos, equipos y/o herramientas que disponga para realizar la formación. Y una vez finalizado el contrato reintegrar y/o devolver los elementos y bienes que, a manera de inventario, el Centro de Formación ponga a su disposición, uso y custodia para la cabal ejecución del contrato. 17. Inscribirse y Certificarse en los procesos de certificación de competencias laborales que apliquen a la función instructor. 18. Participen interdisciplinariamente en el desarrollo curricular, (Estructuración del proyecto, planeación pedagógica, Guías de Aprendizaje), así como en la evaluación del aprendizaje. 19. Apoyar en los procesos de pruebas, talleres aptitudinales y/o selección de aprendices.

LUGAR DE EJECUCIÓN: Barranquilla

PLAZO DE EJECUCIÓN: 4 Mes(es) y 27 días.

FECHA DE INICIACIÓN CONTRATO: **03 DE FEBRERO DE 2020**

FECHA DE TERMINACIÓN DEL CONTRATO: **30 DE JUNIO DE 2020**

VIGENCIA DEL CONTRATO: 2020

VALOR DEL CONTRATO: $ 16.063.086

SUPERVISOR: MANUEL GREGORIO HORMECHEA LANCE

IDENTIFICACION DEL SUPERVISOR: C.C 72.206.101 de Barranquilla

SUSPENSIONES Y AMPLIACIONES DE SUSPENSIÓN: SI____ NO_(X)__

ACTAS No.	TIEMPO	CAUSALES

BALANCE FINANCIERO:

Valor Total del Contrato	$ 16.063.086
Valor Ejecutado	$ 16.063.086
Saldo Pendiente a Cancelar al Contratista	$
Saldo a Liberar	$ 0

PRODUCTOS O ACTIVIDADES EJECUTADAS:

Prestación de servicios profesionales como instructor para impartir formación profesional titulada y complementaria en el área de teleinformática - análisis y desarrollo de sistemas de información de acuerdo a la programación académica del SENA Centro Nacional Colombo Alemán
Soporte a actividades de formación e investigación prioritarias de la coordinación de Automatización.

OBSERVACIONES DE LA SUPERVISIÓN SOBRE LOS PRODUCTOS O ACTIVIDADES EJECUTADAS

SANCIONES: SI___ NO_(X)__

CUMPLIMIENTO FRENTE A LA OBLIGACIÓN A LA SEGURIDAD SOCIAL: SI_(x)_ NO__

GRADO DE SATISFACCIÓN: EXCELENTE_(X)_ BUENO__ REGULAR__ INSUFICIENTE__

Atentamente,

MANUEL GREGORIO HORMECHEA LANCE

EJERCICIOS RESUELTOS

TI8: EJERCICIOS SOBRE VECTORES

4. Crear un vector de 20 elementos con valores numéricos diferentes. Hallar el mayor valor y su posición y el valor promedio.

Análisis:
Datos de Entrada: n←20, vec[20]
Datos de Salida: mayor, pos, promedio
Proceso:
- Un ciclo para llenar el vector con un ciclo anidado y un condicional que controle que no se repitan los datos.
- Un procedimiento que busca el mayor valor y su posición y el valor promedio.

10	40	70	20	50	60	15	30	28	80	25	35	55	45	5	65	75	12	22	32
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

mayor ← 80 posicion← 9 promedio ← 38.7

Algoritmo:

```
Inicio
    Real: vector[20]
    Entero: N←20
    Inicializar (vector, N)  //Este procedimiento llenaría el vector de ceros.
    Calculos (vector, N)
Fin

Procedimiento Inicializar (Real: vec[], Entero: n)
    Entero: i
    Para (i=0, N-1, 1) haga
            vec[i]←0
    Fpara
Fin_Procedimiento

Procedimiento Calculos (Real: vec[], Entero: n)
    Entero: con, i, pos, ban ←1
    Real: mayor, promedio
    Escriba:"Digite el primer valor: "
    Lea: vec[0]
    promedio ← mayor ← vec[0]

    Para (con=1, n-1, 1) haga
            Escriba:"Digite un valor: "
            Lea: vec[con]
            ban ← 1
(1)
```

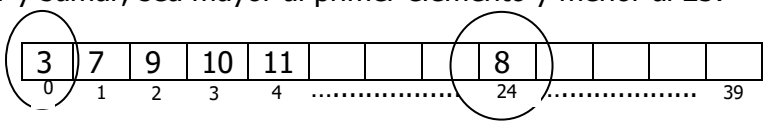
```
1
        Para (i=0, con-1, 1) haga
                Si (vec[con] = vec[i]) ent
                        i ← con
                        ban ← 0
                Fsi
         Fpara
        Si (ban = 1) ent
                promedio ← promedio + vec[con]

                Si (vec[con]>mayor)ent
                        mayor← vec[con]
                        pos ← con
                Fsi
        sino
                Escriba: "El valor ingresado está repetido. Por favor ingrese
                otro diferente"
                con←con-1
        Fsi
Fpara

   promedio ← promedio / n
   Escriba: "El valor mayor es", mayor, "y está en la posición: ",pos
   Escriba: "El valor promedio es", promedio
FinProcedimiento
```

7. En un vector de 40 elementos numéricos llamado totales, se requiere ir sumando y mostrando cada elemento, siempre y cuando sea mayor al primer elemento y menor al elemento 25. Finalmente, muestre la suma.

Análisis:
Datos de Entrada: totales[40] (ya no los dan), n←40
Datos de Salida: suma
Proceso:
- Una función que devuelva la suma.
- Un ciclo para recorrer el vector y una condición que controle que cada elemento a mostrar y sumar, sea mayor al primer elemento y menor al 25.



Por ejemplo, comparando el segundo elemento sería:
Si (7>3) ∧ (7<8) ent
    acumulador←acumulador+7
Fsi

Algoritmo:
Para los casos en que los vectores ya no los den llenos, solo hacemos el subprograma que haga lo que pide el problema y asumimos que a ese subprograma, nos envían el vector y el número de elementos.

```
Funcion sumatotal (Real: vec[], Entero: n)
        Entero: i
        Real: suma ← 0

        Escriba: "Los elementos mayores a", vec[0], "y menores a", vec[24], "son:"

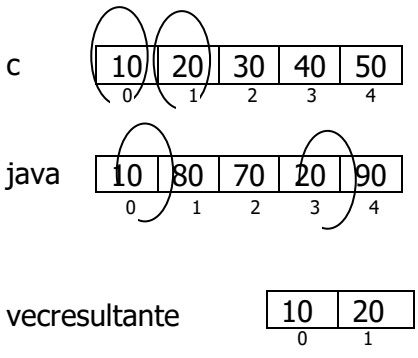
        Para (i=1, n-1, 1) haga
            Si (vec[i] > vec[0]) ∧ (vec[i] < vec[24]) ent
                    suma ← suma + vec[i]
                    Escriba: vec[i]
            Fsi
        Fpara

        Retorne suma
 Fin_Funcion
```

10. Cargar 2 vectores, uno con los códigos de los estudiantes que perdieron C, el segundo con los códigos de los estudiantes que perdieron Java. Se pide crear otro arreglo formado por los códigos de los estudiantes que perdieron ambas materias. Mostrar el arreglo resultante.

Análisis:
Datos de Entrada: c[], nc, java[], nj
Datos de Salida: vecresultante[]
Proceso:

- Un procedimiento para llenar o cargar los vectores, al que se llama dos veces (una para los de C y otra para los de Java).
- Otro procedimiento para llenar un vector resultante con los estudiantes que perdieron las dos materias y luego mostrarlo.
- Dentro de este último procedimiento, se debe comparar un elemento de un vector con todos los elementos del otro vector, para chequear que coincidan.



Algoritmo:

```
Inicio
        Entero: nc, nj
        Real: c[100], java[100]

        Inicializar (c, 100)
        Inicializar (java,100)

        Escriba:"Digite el número de estudiantes que perdieron C"
        Lea: nc
        Escriba: "Ahora digite uno por uno, los códigos de los", nc, "estudiantes que
        perdieron C"
        Llenar (c, nc)

        Escriba:"Digite el número de estudiantes que perdieron Java"
        Lea:nj
        Escriba: "Ahora digite uno por uno, los códigos de los", nj, "estudiantes que
        perdieron Java"
        Llenar (java, nj)

        Resultado (c, java, nc, nj)
Fin

Procedimiento Inicializar (Real: vec[], Entero: n)
    Entero: i
    Para (i=0, N-1, 1) haga
            vec[i]←0
    Fpara
 Fin_Procedimiento


Procedimiento Llenar (Real: vec[], Entero: n)
        Entero: i
        Para (i=0, n-1, 1) haga
            Escriba: "Digite el código", i+1
            Lea: vec[i]
        Fpara
 Fin_Procedimiento


 Procedimiento Resultado (Real: vectorc[], vectorj[], Entero: NC, NJ)
        Real: vecresultante[100]
        Entero: i, j, conv←0
```

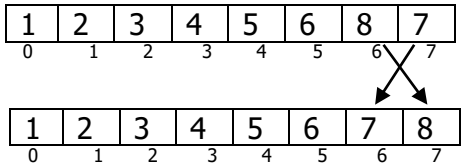
```
Para (i=0, NC-1, 1) haga
        Para (j=0, NJ-1, 1) haga
                Si (vectorc[i] = vectorj[j])
                        vecresultante[conv]← vectorc[i]
                        conv←conv+1
                Fsi
        Fpara
Fpara

    Si (conv=0) ent
        Escriba:" Ningún estudiante perdió las dos materias"
    sino
        Escriba: "Los códigos de los estudiantes que perdieron las dos materias
        fueron:"
        Para(i=0, conv-1, 1)
                Escriba: vecresultante[i]
        Fpara
    Fsi
Fin Procedimiento
```

13. Elabore un algoritmo que lea un vector de N elementos. Si el número de elementos es par intercambiar, partiendo de la última posición, el elemento i-ésimo por el elemento i-ésimo – 1 y así hasta el primer elemento. Mostrar el vector final. Si el número de elementos es impar, indicar en un mensaje que no es posible hacer el intercambio y forzar al usuario hasta que digite un número de elementos par.

Análisis:
Datos de Entrada: N, vector[]
Datos de Salida: vector[]
Proceso:

- Un ciclo para controlar que el usuario solo ingrese un número de elementos par.
- Un procedimiento para intercambiar el elemento de la última posición con el elemento de la penúltima posición y así sucesivamente decrementando.



Vector Resultante:

8	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

Algoritmo:

```
Inicio
        Entero: N
        Real: vector [100]
        Caracter: ban←'s'

        Inicializar (vector, 100)

    Mq(ban='s')
        Escriba: "Digite el número de elementos a guardar en el vector"
        Lea: N
      Si (N%2=0) ent
                Llenar (vector, N)
                Intercambio (vector, N)
                ban←'n'
      sino

                Escriba: "No es posible hacer el intercambio por favor ingrese un
                        número de elementos par"
      Fsi
    FMq
Fin

Procedimiento Inicializar (Real: vec[], Entero: n)
    Entero: i
   Para (i=0, N-1, 1) haga
            vec[i]←0
   Fpara
 Fin_Procedimiento

 Procedimiento Llenar (Real: vec[], Entero: n)
        Entero: i
        Para (i=0, n-1, 1) haga
            Escriba: "Digite el elemento", i+1
            Lea: vec[i]
        Fpara
Fin_Procedimiento

Procedimiento Intercambio (Real: vec[], Entero: n)
        Entero: i
        Real: aux
        Para (i=n-1, 1, -1) haga
            aux ← vec[i]
            vec[i]←vec[i-1]
            vec[i-1]←aux
        Fpara
```

```
        Escriba:"El vector final es:"
        Para (i=0, n-1, 1) haga
            Escriba: vec[i]
        Fpara
Fin_Procedimiento
```

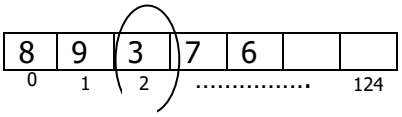
15. Se tiene un vector de 125 elementos con valores numéricos enteros, realice lo siguiente:

- Leer un valor x y buscar en qué posición del vector se encuentra.
- Llene otro vector con los elementos de las posiciones impares del vector dado.
- En este último vector, busque cuántos elementos son múltiplos de 3 y positivos.

Análisis:

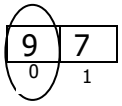
Datos de Entrada: n←125, vec[125], x
Datos de Salida: x, posx, mulypos3
Proceso:

- Un procedimiento para hallar la posición del valor x y mostrar ambos.
- Una función para llenar al vector con los elementos de las posiciones impares y que retorne el número de múltiplos de 3 y positivos (en este punto, se debe tener en cuenta que el módulo representa un número entero).



x←3 posx←2

vectorimpar



mulypos3 ← 1

Algoritmo:
Como ya nos dan el vector lleno, solo hacemos los respectivos subprogramas que haga lo que pide el problema y asumimos que a esos subprogramas, nos envían el vector y el número de elementos.

```
Procedimiento Buscar (Entero: vec[], Entero: n)
    Entero: i←0, x
    Escriba: "Digite un valor para buscarlo y decirle dónde está en el vector"
    Lea: x
        Mq (i<n) ∧ (x <> vec[i]) haga
            i← i + 1
    FMq
```

```
        Si (i<n) ent
                Escriba: "El valor", x, "está en la posición", i
        sino
                Escriba: "El valor no se encuentra en el grupo de datos"
        Fsi
FinProcedimiento

Funcion Llenarposimpar (Entero: vec[], Entero: n)

    Entero: i, j ← 0, mulypos3←0, vectorimpar[n/2]

    Para (i=1,n-1,2) haga
        vectorimpar [j]←vec[i]
        j←j+1
    Fpara

    Para (i=0, j-1,1) haga
        Si (vectorimpar [i]%3=0) ∧ (vectorimpar [i]>0) ent
            mulypos3←mulypos3+1
        Fsi
    Fpara

    Retorne mulypos3
FinFuncion
```



Laboratorio

Fundamentos de programación con LPP

FAVA - Formación en Ambientes Virtuales de Aprendizaje

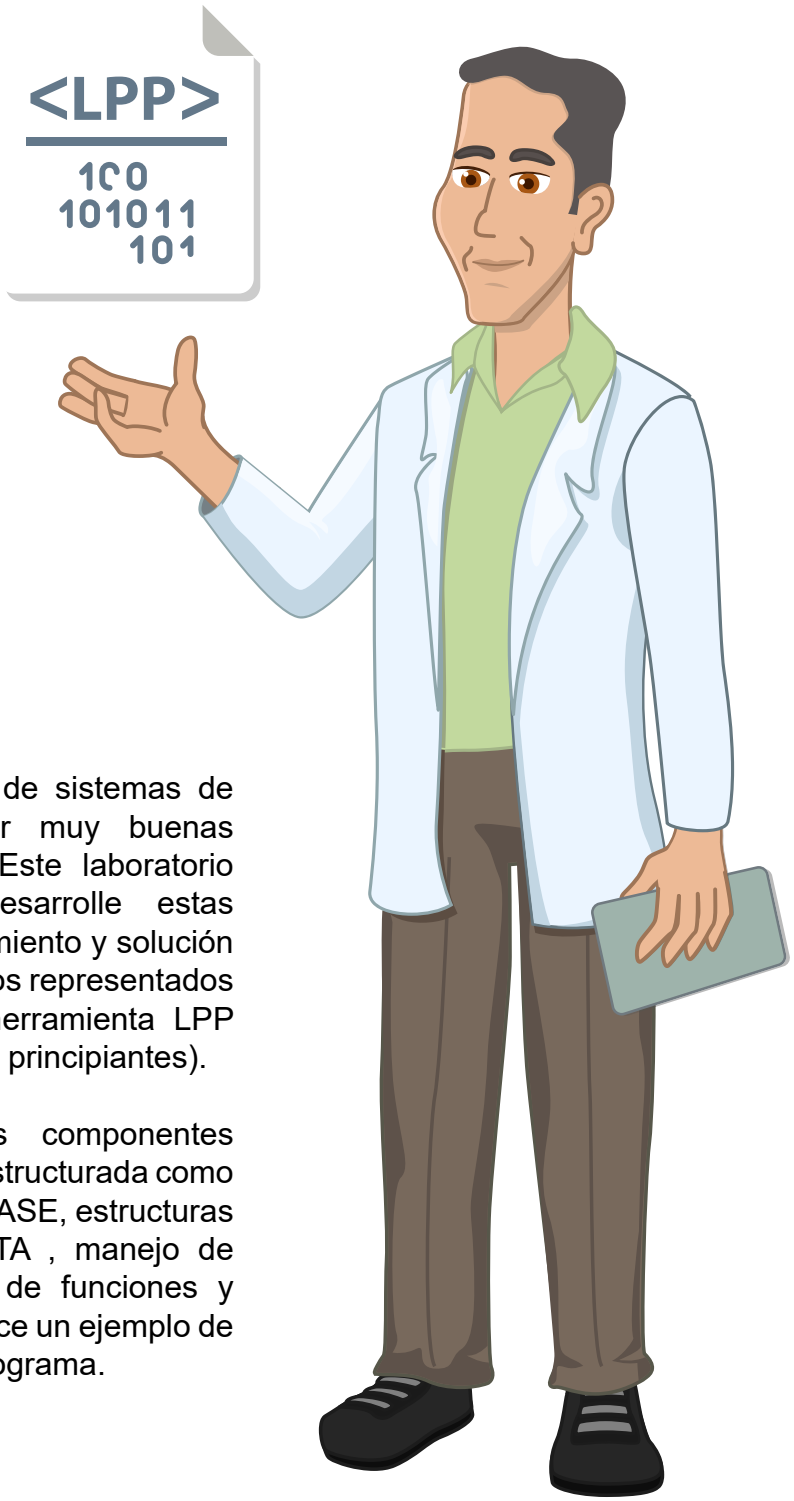
SENA - Servicio Nacional de Aprendizaje.

ESTRUCTURA DE CONTENIDOS

Pág.

Introducción 3
1. Objetivos 4
2. Consideraciones 4
3. Procedimiento 5
3.1. Pasos 5
3.2. Pasos para desarrollar un algoritmo mediante diagramas de flujo de datos 6
3.3. Algoritmos a desarrollar 7
4. Evidencias 8
Glosario 10
Bibliografía 11
Control del documento 12

FUNDAMENTOS DE PROGRAMACIÓN CON LPP



Introducción

Para el diseño y el desarrollo de sistemas de información se requiere tener muy buenas habilidades de programación. Este laboratorio permite que el aprendiz desarrolle estas habilidades mediante el planteamiento y solución de problemas mediante algoritmos representados en pseudo-código usando la herramienta LPP (Lenguaje de programación para principiantes).

Este recurso comprende los componentes principales de la programación estructurada como son: estructuras de control: SI, CASE, estructuras cíclicas: PARA y REPITA-HASTA , manejo de variables tipo arreglo, manejo de funciones y procedimientos. Por último se hace un ejemplo de la revisión paso a paso de un programa.

DESARROLLO DE CONTENIDOS

1. Objetivos

OBJETIVO GENERAL

Diseñar, construir e implementar algoritmos expresados en seudocódigo.



OBJETIVOS ESPECÍFICOS

- Identificar los pasos a seguir en la construcción de algoritmos.
- Diferenciar la utilidad de cada uno de los elementos del lenguaje proporcionados por LPP.
- Aplicar las estructuras de programación de acuerdo con la situación presentada.
- Utilizar el programa LPP para el desarrollo de los algoritmos planteados.

2. Consideraciones

Para el desarrollo del laboratorio es importante tener en cuenta los siguientes aspectos relacionados en la tabla:

ÍTEM	DESCRIPCIÓN
Soporte Teórico	• Revisar los Objetos de Contenido sobre Fundamentos de Programación Estructurada y Estructuras Cíclicas. • Revisar los video tutoriales que acompañan este laboratorio.
Productos requeridos	• Ejercicios ejemplo que se muestran en los objetos de contenido y en los Video tutoriales. • Lista de los algoritmos a desarrollar en este laboratorio (No calificables).
Herramientas SW	• Para el desarrollo del Laboratorio se requiere tener instalada la Herramienta LPP de acuerdo con las instrucciones suministradas en el video-tutorial que acompaña este recurso.

3. Procedimiento



Para el desarrollo de cada uno de los ejercicios tener en cuenta el siguiente procedimiento:

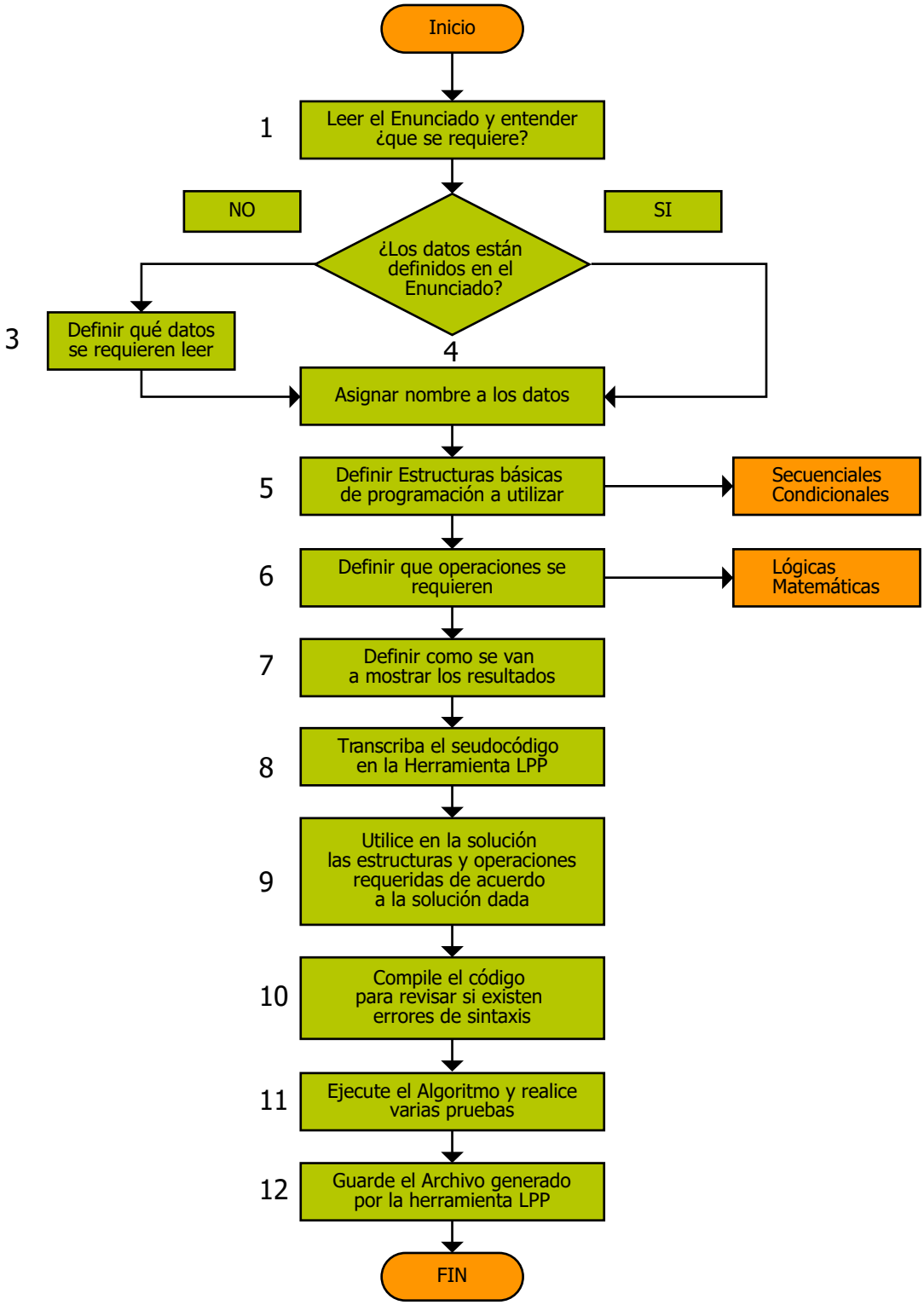
Para iniciar se requiere primero planificar la estructura de solución en una hoja de papel y después hacerlo utilizando la herramienta LPP.

3.1. Pasos

Se recomienda seguir activamente las siguientes instrucciones:

- Leer detenidamente y por completo el enunciado, hasta poder identificar claramente qué es lo que se está solicitando.
- Revisar si todos los datos necesarios para resolverlo están definidos en el enunciado o si se requiere leer otros datos.
- Si ya hay datos definidos asígnarles un nombre a cada uno de estos datos.
- Si se requiere solicitar datos (leer datos), definir qué datos se van a solicitar, el orden y colócarles un nombre a cada dato a leer.
- Pregúntarse ¿Qué operaciones (fórmulas) se debe utilizar para obtener los resultados?
- Pregúntarse ¿Qué estructuras se debe utilizar para obtener los resultados?
  - Estructuras secuenciales
  - Estructuras condicionales
  - Estructuras repetitivas.
- Definir cómo va a presentar (mostrar) los resultados.
- Transcribir el seudocódigo en la herramienta LPP, utilizando las estructuras y operaciones requeridas de acuerdo a la solución dada.
- Compilar el código para revisar si existen errores de sintaxis.
- Ejecutar el algoritmo y realizar varias pruebas.
- Guardar el archivo en una carpeta para después ser enviados al tutor.

3.2. Pasos para desarrollar un algoritmo mediante diagramas de flujo de datos



3.3. Algoritmos a desarrollar

Diseñar y construir los algoritmos expresados mediante seudocódigo utilizando la herramienta software LPP, a partir de los siguientes enunciados:

PRIMER ENUCIADO: desarrollar un algoritmo que calcule el salario neto que debe recibir un vendedor de un almacén. Se debe tener en cuenta si tiene derecho o no al auxilio de transporte. Para el desarrollo del ejercicio tenga en cuenta las siguientes formulas:

Sueldo devengado = salario básico * días laborados / 30.

Días laborados = debe ser entre 1 y 30.

Auxilio de Transporte: Lo reciben los empleados cuyo salario básico sea menor o igual a 2 salarios mínimos legales vigentes.

Salario Mínimo Legal Vigente(2017): 737.717

Auxilio de Transporte = 83.140 * días laborados / 30 (año 2017 en Colombia).

Comisión de Ventas: En la empresa se tiene estipulado dar una comisión de ventas del 2% sobre las ventas del mes de cada vendedor.

Total devengado = sueldo devengado + comisión de ventas.

Total deducciones = descuentos por prestamos.

Salario Neto = Total devengado – Total deducciones

Como resultado del ejercicio se debe imprimir en pantalla lo siguiente:

Cedula empleado: XXXXXX

Nombre Empleado: XXXXXXX

Salario Básico: XXXXXX

Auxilio de Transporte: XXXXXX

Comisión de Ventas: XXXXXX

Préstamos: XXXXXX

Salario Neto a Recibir: XXXXX

SEGUNDO ENUNCIADO: hacer un algoritmo que imprima los primeros 20 términos de la siguiente serie:

1, 3, 6, 10, 15, 21, 28,……..

TERCER ENUNCIADO: desarrollar un algoritmo que permita calcular los siguientes datos de una fiesta:

¿Cuántas personas asistieron a la fiesta?

¿Cuántos hombres y cuantas mujeres?

- Promedio de edades por sexo.
- La edad de la persona más joven que asistió.

Consideraciones:

- No se permiten menores de edad a la fiesta.
- Ingresar datos hasta que se ingrese una edad igual a cero.

CUARTO ENUNCIADO: hacer un algoritmo que imprima el costo de una llamada telefónica, capturando la duración de la llamada en minutos y conociendo lo siguiente:

- Toda llamada que dure tres minutos o menos tiene un costo de $200.
- Cada minuto adicional cuesta $30.

QUINTO ENUNCIADO: un grupo de 10 estudiantes presentan un examen de Física. Hacer un algoritmo que lea por cada estudiante la calificación obtenida. Al finalizar calcule e imprima:

- La cantidad de estudiantes que obtuvieron una calificación menor a 50.
- La cantidad de estudiantes que obtuvieron una calificación de 50 o más pero menor que 70.
- La cantidad de estudiantes que obtuvieron una calificación de 70 o más pero menor que 80.
- La cantidad de estudiantes que obtuvieron una calificación de 80 o más.

La calificación obtenida en el examen de física debe ser entre 1 y 100.

4. Evidencias



- Archivo comprimido en formato .zip con los 5 archivos fuentes de cada uno de los ejercicios realizados en la herramienta LPP.
- En un documento, mostrar evidencias mediante pantallazos de la ejecución de cada uno de los ejercicios.

NOTA:
La realización de los ejercicios en este laboratorio, tiene como finalidad afianzar los conocimientos adquiridos y desarrollar mayor comprensión y práctica para alcanzar los resultados de aprendizaje planteados en esta Actividad de proyecto, por tal motivo no son actividades calificables.

Glosario

Seudocódigo: es una descripción de alto nivel de un algoritmo utilizando lenguaje natural y algunas convenciones sintácticas propias de los lenguajes de programación.

LPP: acrónimo Lenguaje de programación para principiantes. Software para la enseñanza de programación básica.

Bibliografía

Castillo, R. (2001). Programación en LPP. Honduras.

Control del documento

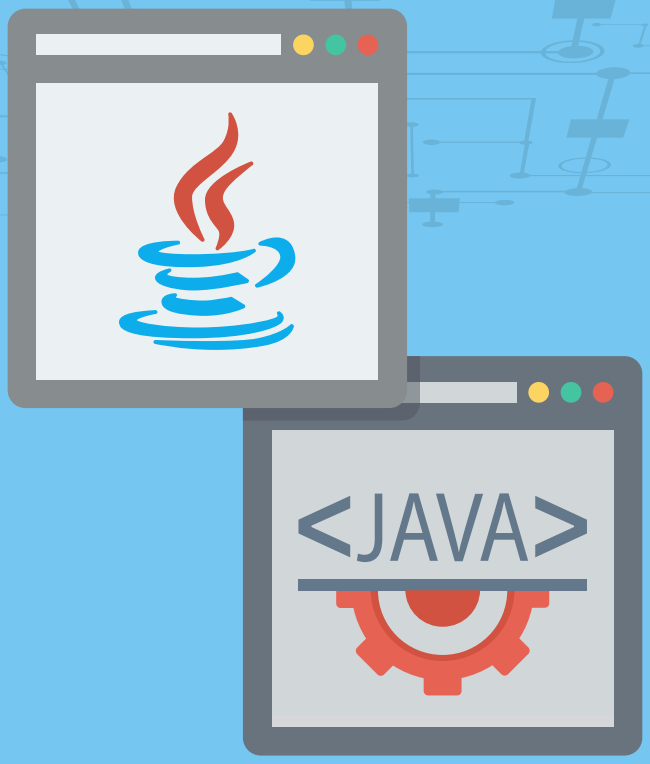
CONSTRUCCIÓN OBJETO DE APRENDIZAJE

LABORATORIO. FUNDAMENTOS DE PROGRAMACIÓN CON LPP	
Centro Industrial de Mantenimiento Integral - CIMI Regional Santander	
Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Experto temático:	César Marino Cuéllar Chacón (V1) Nelson Mauricio Silva M. (V2)
Diseño multimedia:	Luis Gabriel Urueta Alvarez
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño

creative commons

BY NC SA

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de la licencia que el trabajo original.



Laboratorio

Programación con el lenguaje Java

FAVA - Formación en Ambientes Virtuales de Aprendizaje

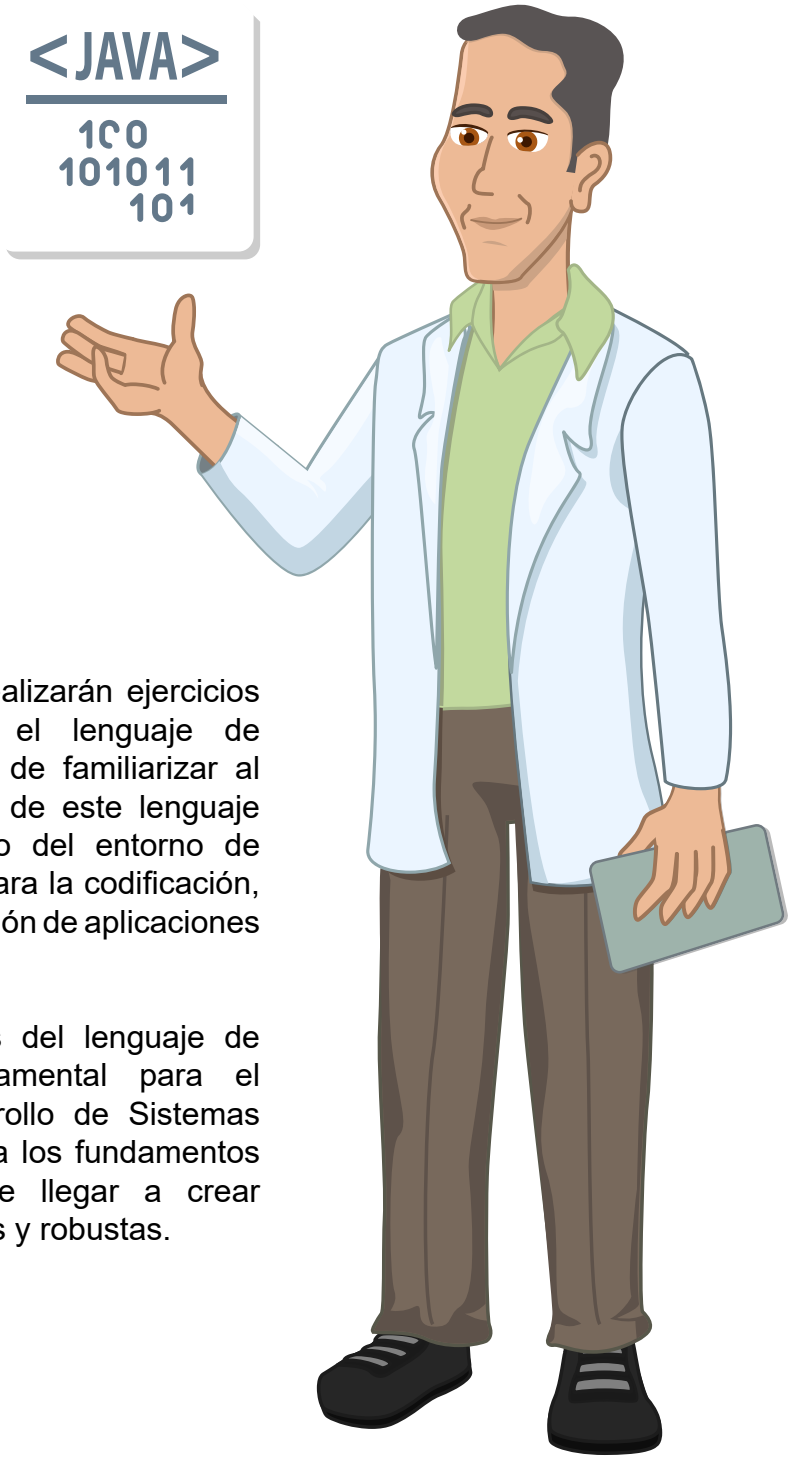
SENA - Servicio Nacional de Aprendizaje.

Estructura de contenidos

Pág.

Introducción 3
1. Objetivos 4
2. Consideraciones 4
3. Procedimiento 5
3.1. Pasos 5
3.2. Pasos para desarrollar un algoritmo mediante diagramas de flujo de datos 9
3.3. Algoritmos a desarrollar 10
4. Evidencias 14
Glosario 15
Bibliografía 16
Control del documento 17

PROGRAMACIÓN CON EL LENGUAJE JAVA



Introducción

En el presente laboratorio se realizarán ejercicios de programación empleando el lenguaje de programación JAVA, con el fin de familiarizar al aprendiz con la sintaxis propia de este lenguaje de programación y con el uso del entorno de desarrollo integrado Netbeans para la codificación, compilación, depuración y ejecución de aplicaciones JAVA.

Practicar y apropiar la sintaxis del lenguaje de programación JAVA es fundamental para el tecnólogo en Análisis y Desarrollo de Sistemas de Información, porque le brinda los fundamentos necesarios para posteriormente llegar a crear soluciones de software integrales y robustas.

Desarrollo de contenidos

1. Objetivos



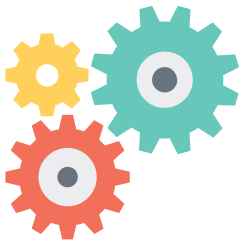
- Emplear el entorno de desarrollo integrado NETBEANS para la codificación, compilación, depuración y ejecución de aplicaciones JAVA.
- Declarar variables empleando los tipos de datos propios de JAVA, de acuerdo con los requisitos de las aplicaciones a desarrollar.
- Aplicar la sintaxis del lenguaje de programación JAVA en el uso de las estructuras de programación secuenciales, condicionales y cíclicas.
- Crear aplicaciones en JAVA que respondan a los requerimientos funcionales solicitados, aplicando la lógica de programación necesaria y las características técnicas del lenguaje de programación JAVA.

2. Consideraciones

Para el desarrollo del laboratorio es importante que tenga en cuenta los siguientes aspectos relacionados en la tabla:

ÍTEM	DESCRIPCIÓN
Soporte Teórico	• Laboratorio. Construir algoritmos con la herramienta DFD. • Laboratorio. Construir algoritmos con la herramienta LPP. • Introducción al lenguaje de programación JAVA.
Productos requeridos	• Ejercicios Laboratorio Construir algoritmos con la herramienta DFD. • Ejercicios Laboratorio Construir algoritmos con la herramienta LPP.
Herramientas SW	• Netbeans, de acuerdo con las instrucciones del videotutorial enlace de descarga.

3. Procedimiento



Para el desarrollo de cada uno de los ejercicios se debe tener en cuenta el siguiente procedimiento:

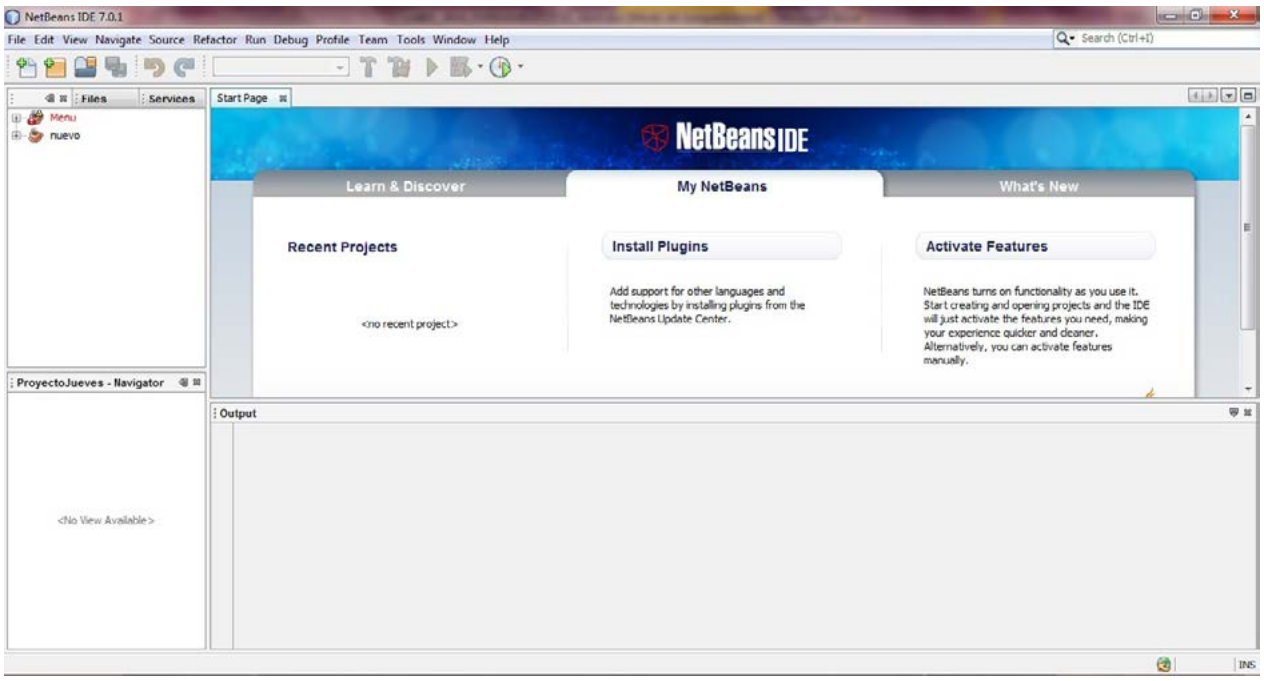
Los ejercicios a codificar en este laboratorio son los ejercicios planteados en el laboratorio de construir algoritmos con la herramienta DFD y el laboratorio construir algoritmos con la herramienta LPP, por lo tanto, el énfasis del presente laboratorio se basa principalmente en aplicar la sintaxis del lenguaje JAVA sobre soluciones algorítmicas comprobadas.

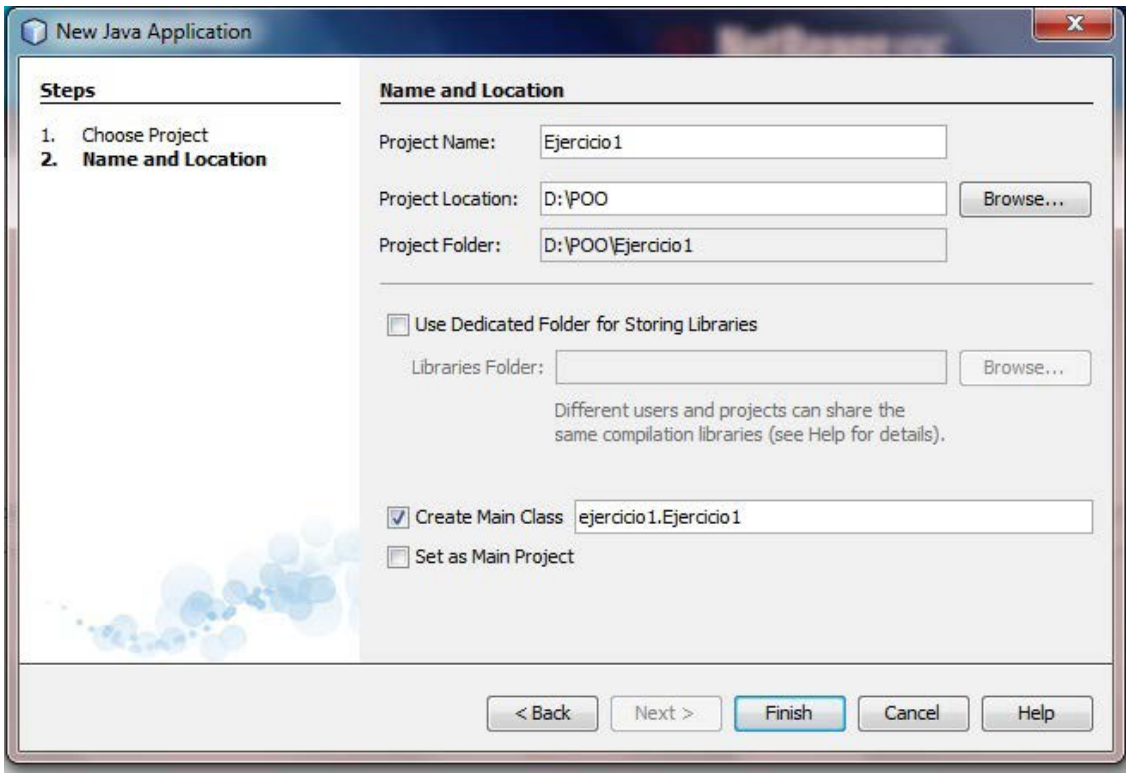
3.1. Pasos

Se recomienda seguir activamente las siguientes instrucciones:

Leer detenidamente y por completo el enunciado de cada ejercicio y comprobar la solución del mismo realizada en el laboratorio construir algoritmos con la herramienta DFD o el laboratorio construir algoritmos con la herramienta LPP.

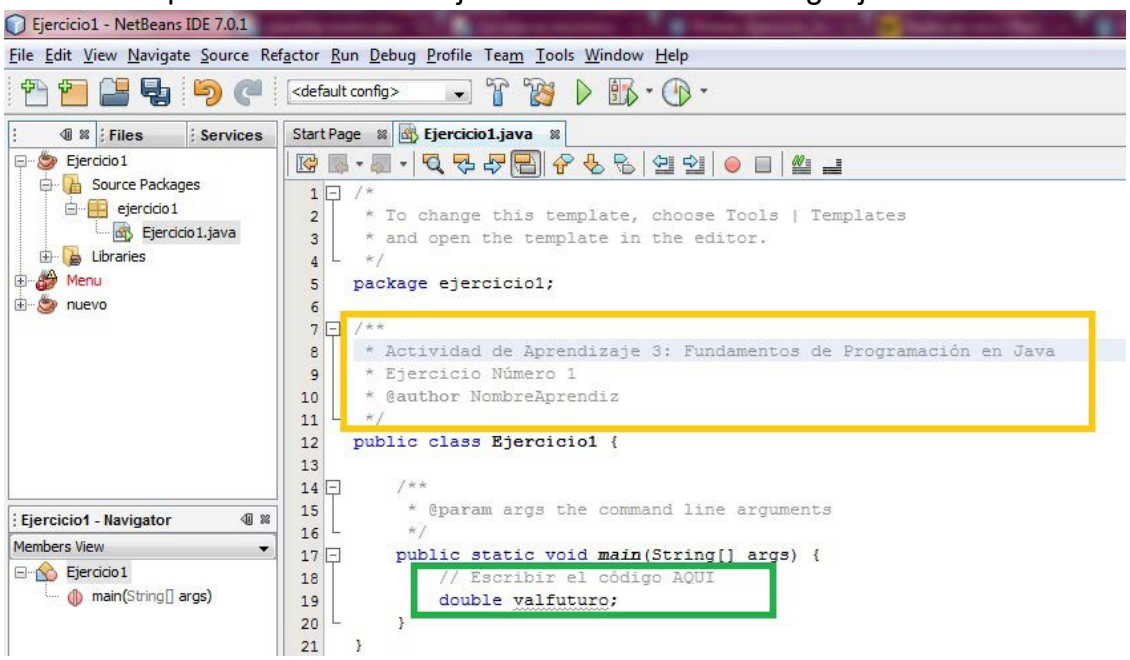
Realizar la codificación de cada ejercicio en el lenguaje de programación JAVA, empleando la herramienta Netbeans. Para esto es necesario haber consultado el objeto de contenido “Introducción al lenguaje de programación JAVA” y los videotutoriales de instalación y uso del entorno de desarrollo.

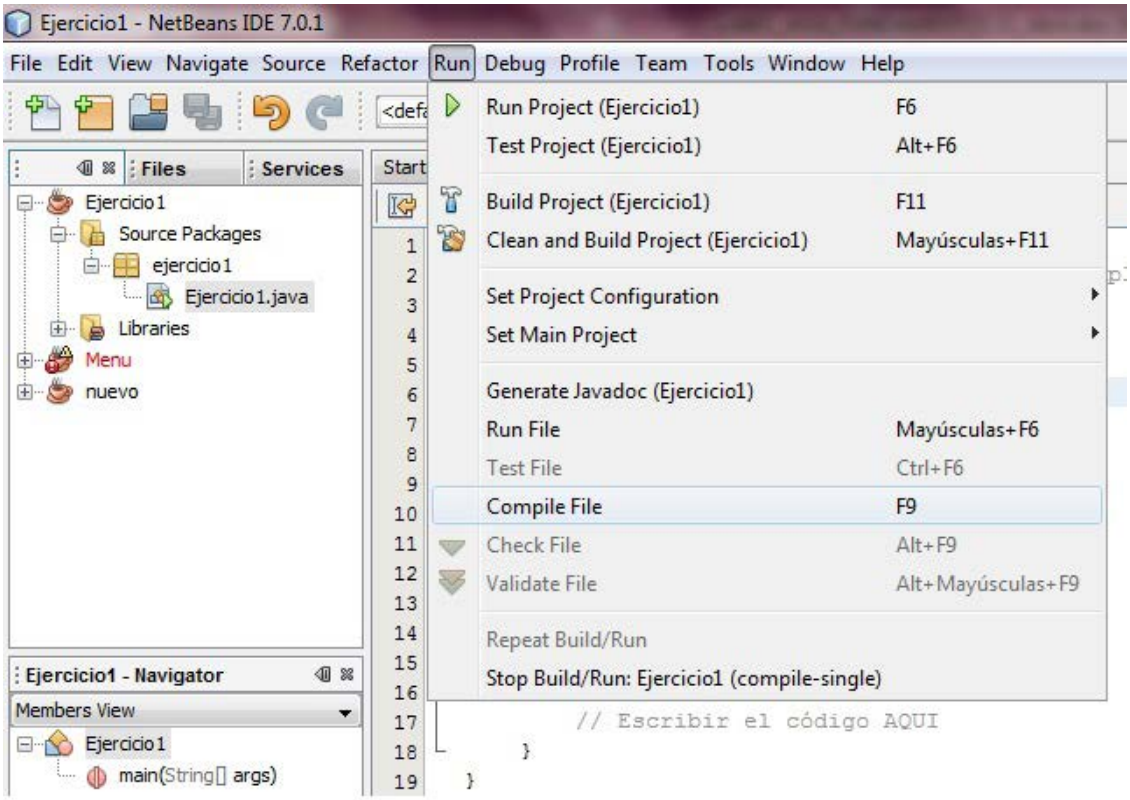




Crear la aplicación con el nombre EjercicioX para cada uno de los ejercicios.

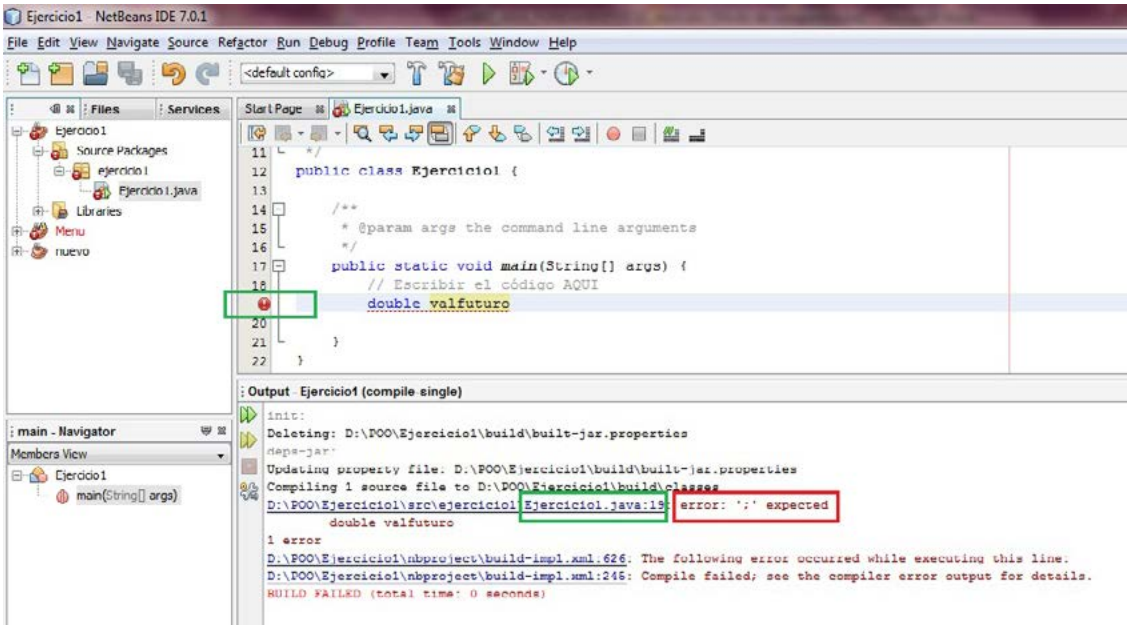
Editar la sección de comentarios, identificando como autor del programa y escribir el algoritmo correspondiente de cada ejercicio codificado en lenguaje JAVA.

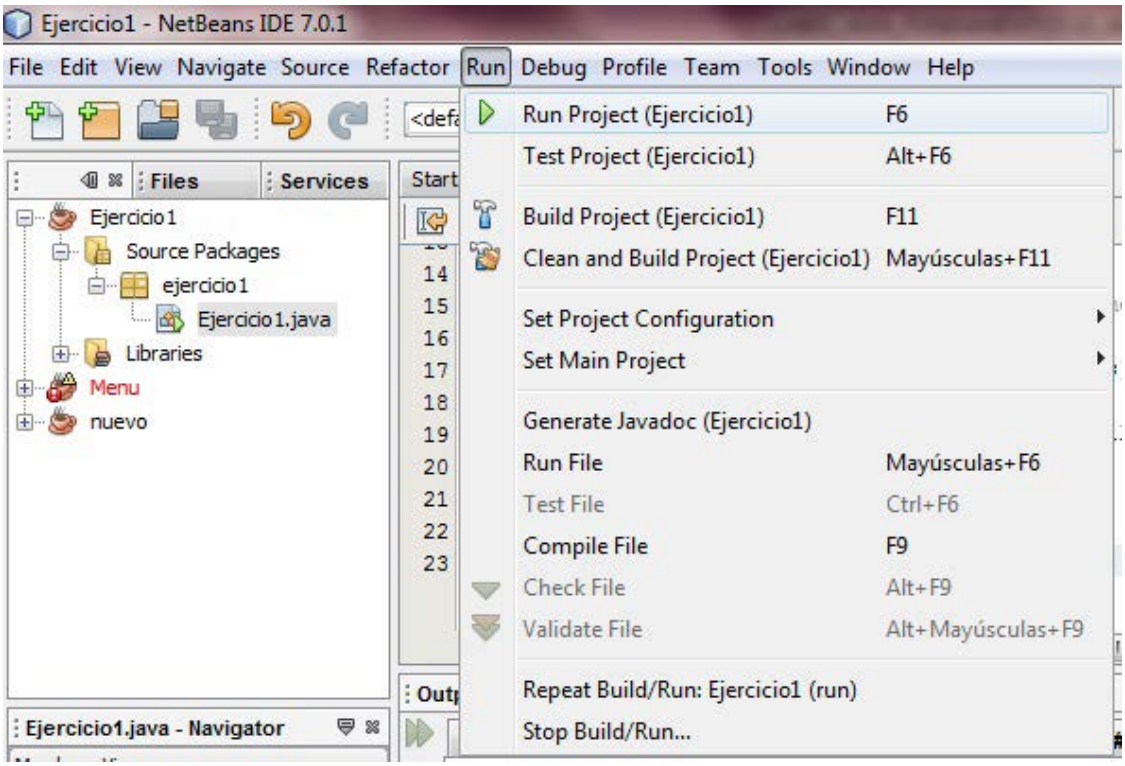




Compilar el código escrito en NETBEANS, presionando la tecla F9.

En caso de presentar errores de compilación, regresar al paso 2 y corregir los errores.





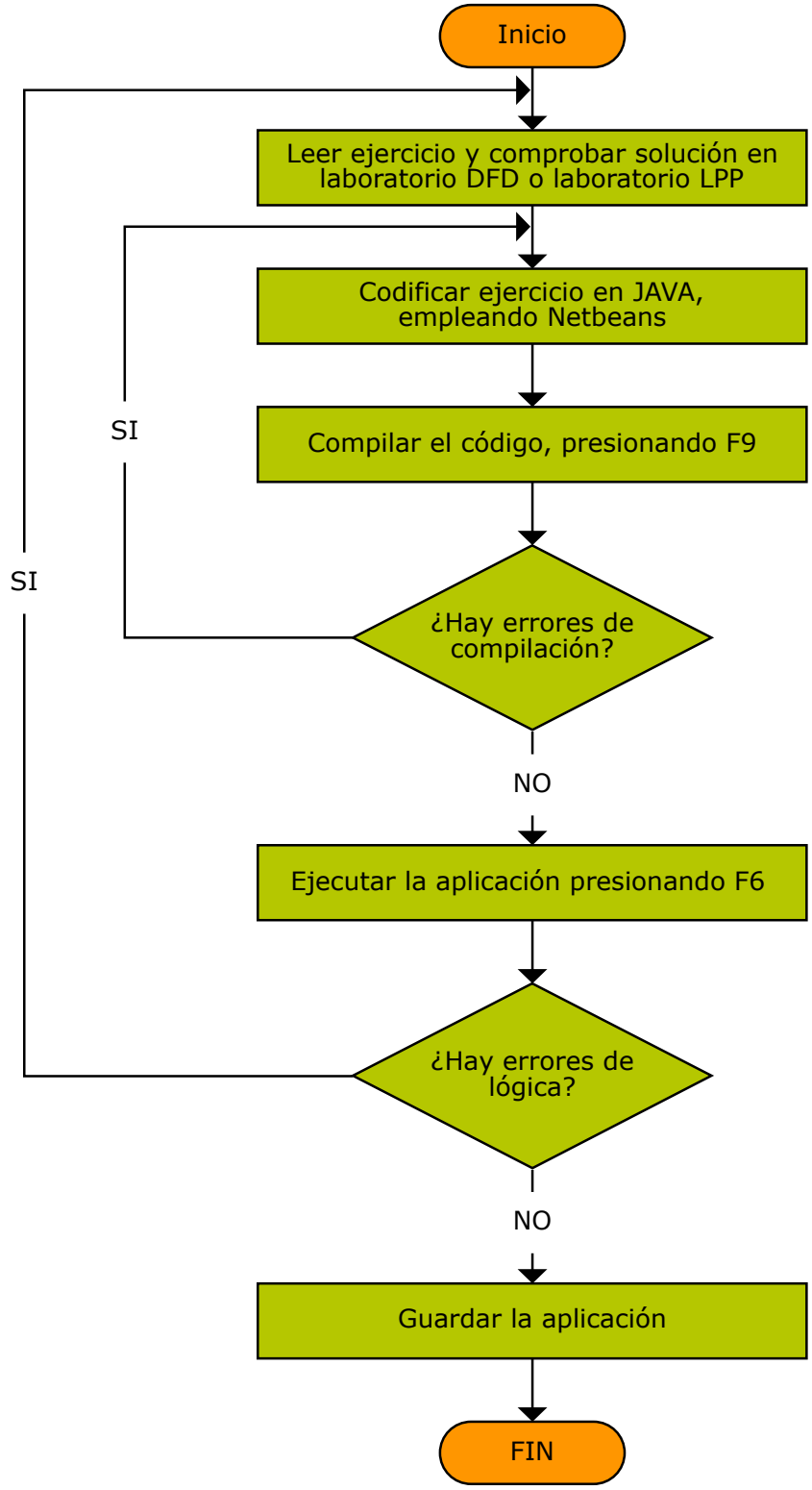
Ejecutar la aplicación, presionando la tecla F6 o utilizando la opción del menú Ejecutar.

En caso de encontrarse errores de lógica en el programa, se regresar al paso 1 y se revisa la solución presentada en DFD o LPP.

Cada ejercicio deberá ser guardado en un archivo aparte con el nombre del respectivo ejercicio, por ejemplo, para el primer ejercicio identificar el archivo como ejercicio1.java.

El siguiente diagrama representa el procedimiento general del laboratorio.

3.2. Pasos para desarrollar un algoritmo mediante diagramas de flujo de datos



3.3. Algoritmos a desarrollar

Nota: recordar que cada uno de los siguientes ejercicios ya fue resuelto en el laboratorio de DFD o el laboratorio de LPP, por lo tanto, el trabajo en este laboratorio consiste en codificar las soluciones en JAVA, siguiendo el procedimiento descrito en el numeral 4 del presente laboratorio.

PRIMER ENUNCIADO: una persona deposita hoy al Banco cierta cantidad de dinero, donde le reconocen un interés del 2% mensual, capitalizado mensualmente. ¿Cuál será el saldo al cabo de 5 años?

Formula a utilizar:
*Valor Futuro = Valor Presente * (1+intereses)^meses*

Para solucionar el Ejercicio, se requiere desarrollar un algoritmo que permita conocer a una persona ¿Cuál será el saldo al cabo de 5 años?, si al depositar hoy cierta cantidad de dinero. Como resultado se debe imprimir en pantalla:

Valor consignado: $XXXXX
Valor futuro en 5 años: $XXXXXXX

Presentar la solución a este ejercicio en JAVA.

SEGUNDO ENUNCIADO: una frutería ofrece las manzanas a $4.200 el kilo, con un descuento de acuerdo a la siguiente tabla:

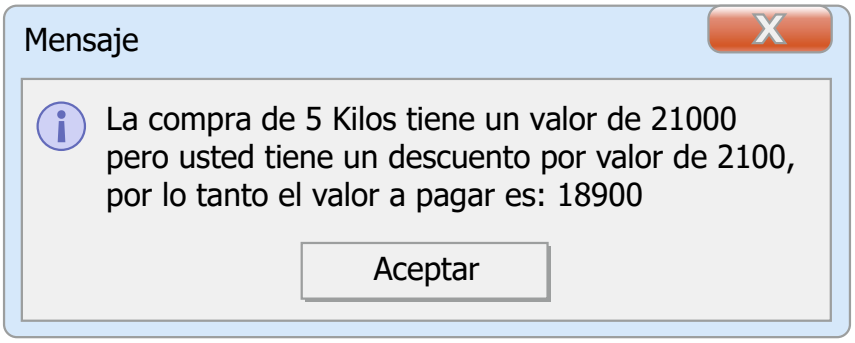
NUMERO DE KILOS%	DE DESCUENTO
0-2	0%
2,01 - 5	10%
5,01 - 10	15%
10.01 EN ADELANTE	20%

Desarrollar un algoritmo que permita a la frutería y al cliente conocer cuanto pagará un cliente que compre manzanas.

Mostrar los resultados así:
La compra de N kilos tiene un valor de **$**, pero tiene un descuento por valor de **$**, por lo tanto el valor a pagar es: **$**.

EJEMPLO:
Si una persona compra 5 kilos de manzanas, el resultado sería:

La compra de 5 kilos tiene un valor de $21000, pero tiene un descuento por valor de $2100, por lo tanto el valor a pagar es: $18900.



Presentar la solución a este ejercicio en JAVA.

TERCER ENUNCIADO: dada la siguiente función:

Y= x^2 – 2x

Se requiere desarrollar un algoritmo que imprima en pantalla, para valores de x desde 1 a 10, lo siguiente:

a. La suma de todos los valores de **Y**.
b. Valores de **Y** múltiplos de 3.
c. Suma de los valores de **Y** múltiplos de 3.
d. Valores de **Y** cuyo último digito sea 5.
e. Suma de los valores de **Y** cuyo último digito es 5.

Presentar la solución a este ejercicio en JAVA.

CUARTO ENUNCIADO: desarrollar un algoritmo que lea un número N entero positivo de cualquier número de dígitos, calcule la suma de sus dígitos y que imprima en pantalla el número leído y la suma de sus dígitos. Se requiere que en el desarrollo utilice la Estructura Cíclica Mientras.

Ejemplo:
N= 3567
SUMA= 21

Presentar la solución a este ejercicio en JAVA.

QUINTO ENUNCIADO: desarrollar un algoritmo que calcule el salario neto que debe recibir un vendedor de un almacén. Se debe tener en cuenta si tiene derecho o no al auxilio de transporte. Para el desarrollo del ejercicio tenga en cuenta las siguientes formulas:

Sueldo devengado = salario básico * días laborados / 30.

Días laborados = debe ser entre 1 y 30.

Auxilio de Transporte: Lo reciben los empleados cuyo salario básico sea menor o igual a 2 salarios mínimos legales vigentes.

Salario Mínimo Legal Vigente(2017): 737.717

Auxilio de Transporte = 83.140 * días laborados / 30 (año 2017 en Colombia).

Comisión de Ventas: En la empresa se tiene estipulado dar una comisión de ventas del 2% sobre las ventas del mes de cada vendedor.

Total devengado = sueldo devengado + comisión de ventas.

Total deducciones = descuentos por prestamos.

Salario Neto = Total devengado – Total deducciones

Como resultado del ejercicio se debe imprimir en pantalla lo siguiente:

Cedula empleado: XXXXXX

Nombre Empleado: XXXXXXX

Salario Básico: XXXXXX

Auxilio de Transporte: XXXXXX

Comisión de Ventas: XXXXXX

Préstamos: XXXXXX

Salario Neto a Recibir: XXXXX

Presentar la solución a este ejercicio en JAVA.

SEXTO ENUNCIADO: hacer un algoritmo que imprima los primeros 20 términos de la siguiente serie:

1, 3, 6, 10, 15, 21, 28,……..

Presentar la solución a este ejercicio en JAVA.

SÉPTIMO ENUNCIADO: desarrollar un algoritmo que permita calcular los siguientes datos de una fiesta:

¿Cuántas personas asistieron a la fiesta?

¿Cuántos hombres y cuantas mujeres?

- Promedio de edades por sexo.
- La edad de la persona más joven que asistió.

Consideraciones:

- No se permiten menores de edad a la fiesta.
- Ingresar datos hasta que se ingrese una edad igual a cero.

Presentar la solución a este ejercicio en JAVA.

OCTAVO ENUNCIADO: hacer un algoritmo que imprima el costo de una llamada telefónica, capturando la duración de la llamada en minutos y conociendo lo siguiente:

- Toda llamada que dure tres minutos o menos tiene un costo de $200.
- Cada minuto adicional cuesta $30.

Presentar la solución a este ejercicio en JAVA.

NOVENO ENUNCIADO: un grupo de 10 estudiantes presentan un examen de Física. Hacer un algoritmo que lea por cada estudiante la calificación obtenida. Al finalizar calcule e imprima:

- La cantidad de estudiantes que obtuvieron una calificación menor a 50.
- La cantidad de estudiantes que obtuvieron una calificación de 50 o más pero menor que 70.
- La cantidad de estudiantes que obtuvieron una calificación de 70 o más pero menor que 80.
- La cantidad de estudiantes que obtuvieron una calificación de 80 o más.
- La calificación obtenida en el examen de física debe ser entre 1 y 100.

Presentar la solución a este ejercicio en JAVA.

4. Evidencias

- Carpeta comprimida con extensión .zip o .rar con el nombre Lab_Java, esta carpeta debe contener cada uno de los ejercicios propuestos en un archivo con extensión .java. La estructura de la carpeta estaría de acuerdo con la siguiente imagen:



NOTA:
La realización de los ejercicios en este laboratorio, tiene como finalidad afianzar los conocimientos adquiridos y desarrollar mayor comprensión y práctica para alcanzar los resultados de aprendizaje planteados en esta Actividad de proyecto, por tal motivo **no son actividades calificables**.

Glosario

Ejecutar, ejecución (run): hacer funcionar un programa instrucción a instrucción.

Flujo (Stream): término que describe el flujo de datos continúo de una dirección entre un emisor y un receptor.

Lenguaje de programación (programming language): notación utilizada por los programadores para escribir programas. un lenguaje tiene una sintaxis (las palabras y símbolos utilizadas para escribir códigos de programa), una gramática (las reglas que definen una secuencia de palabras y símbolos significativos y correctos) y semántica. Java es un lenguaje de programación.

Ligadura estática (static briding): enlace o conexión de un nombre de un método a un cuerpo del método ejecutado por el compilador mediante el análisis léxico del texto de un programa,

Método estático (static method): véase método de una clase. Método declarado en una clave que se llama directamente sin necesidad de que el objeto sea llamado. Variable estática (static variable) Véase variable de clase.

Programa (program): un conjunto de instrucciones (o sentencias) que describen alguna aplicación o actividad ejecutada en una computadora.

Protegido (protected): un modificador para los miembros de una clase. Un miembro protegido de una clase que puede ser utilizado en la clase que está declarad o cualquier subclase derivada de esa clase.

Público (public): un modificador de clases, datos y métodos a los que se puede acceder por todos los programas. Palabra reservada (reserved word) Véase palabra clave.

Sentencia (statement): una unidad de código que representa una acción o una secuencia de acciones. Las sentencias se ejecutan en el orden en que están escritas y siempre terminan en un punto y coma.

Bibliografía

Netbeans. (2017). General Java Development Learning Trail. Recuperado de https://netbeans.org/kb/docs/java/quickstart.html

Control del documento

CONSTRUCCIÓN OBJETO DE APRENDIZAJE

SENA

LABORATORIO. PROGRAMACIÓN CON EL LENGUAJE JAVA	
Centro Industrial de Mantenimiento Integral - CIMI Regional Santander	
Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Experto temático:	Andrés Julian Valencia O. (V1) Edward José Beltrán Lozano (V2)
Diseño multimedia:	Luis Gabriel Urueta Alvarez
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño

creative commons

BY NC SA

Este material puede ser distrubuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismo términos de la licencia que el trabajo original.





INTRODUCCIÓN AL LENGUAJE DE MODELADO UML

FAVA - Formación en Ambientes Virtuales de Aprendizaje

SENA - Servicio Nacional de Aprendizaje.

Estructura de contenidos

Pág.

Introducción 3
Mapa de contenido 4
1. El lenguaje unificado de modelado “UML” 5
1.1 Características de UML 6
2. Tipos de diagramas UML 7
3. Caso de estudio práctico 11
3.1 Casos de uso 12
3.1.1 Componentes 12
3.1.2 Representación gráfica 13
3.1.3 Tipos de relaciones 14
3.1.4 Documentación 15
3.2 Diagrama de secuencia 15
3.2.1 Representación gráfica 16
3.2.2 Tipos de mensajes 16
3.2.3 Diagrama de secuencia <<include>> 17
3.3 Diagrama de comunicación (Colaboración) 18
3.3.1 Representación gráfica 18
3.4 Diagrama de actividades 19
3.4.1 Representación gráfica 20
3.5 Diagrama de clases 23
3.5.1 Clases 23
3.5.2 Relaciones 24
3.6 Diagrama de objetos 27
3.6.1 Representación gráfica 27
3.7 Diagrama de estados 28
3.7.1 Representación gráfica 28
Glosario 31
Bibliografía 32
Control del documento 33

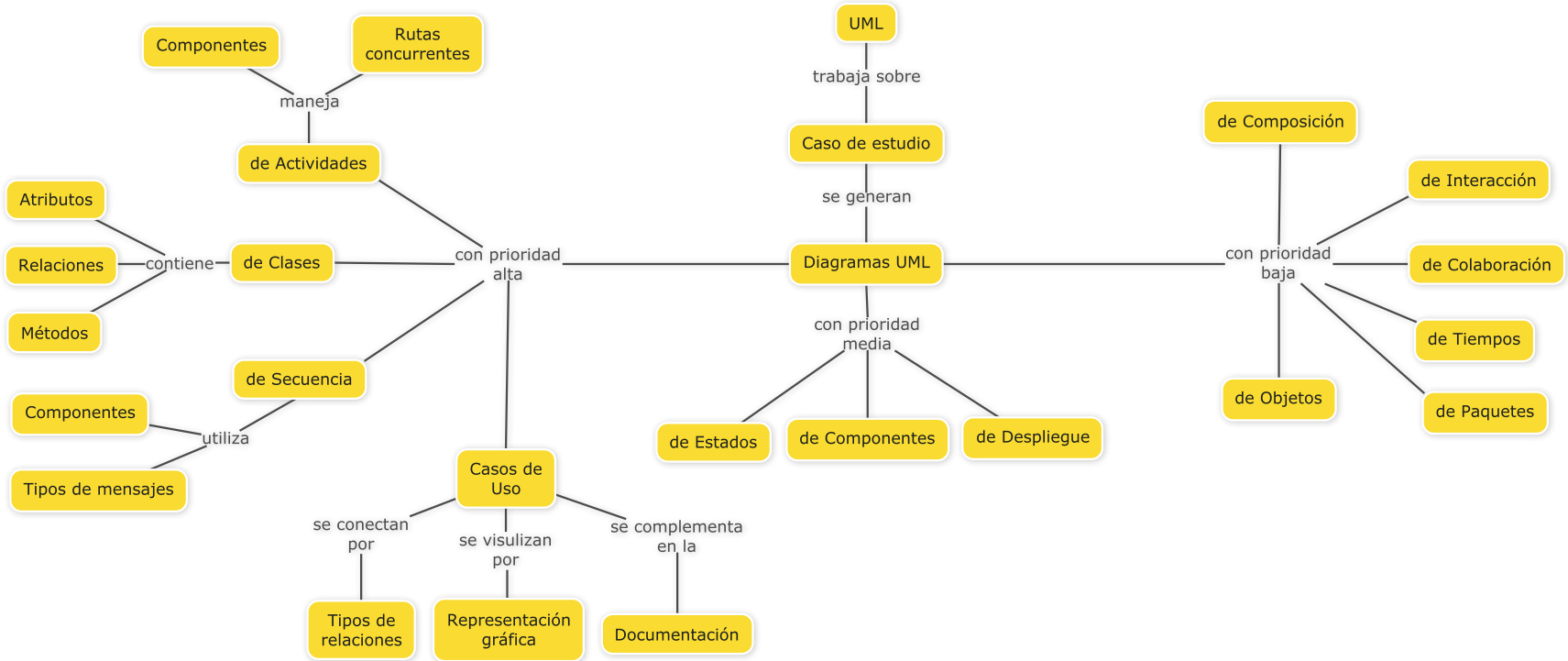
Introducción



Es de suma importancia que para el diseño y desarrollo de Sistemas de Información que el aprendiz en formación como futuro analista y desarrollador de sistemas, conozca los fundamentos del Modelado UML, los tipos de diagramas y su simbología de manera que pueda reconocer su utilidad y reflejar mediante su uso los requerimientos funcionales o necesidades del sistema de información de una forma estandarizada. El lenguaje de modelado UML permita describir los pasos que el actor sigue para realizar una tarea y a su vez, lograr una comunicación adecuada con los demás integrantes del desarrollo del sistema y los clientes, de esta forma, los clientes hacen parte activa en el modelado reduciendo notablemente los correctivos o controles de cambios posteriores.

En este objeto de aprendizaje se describen diversos tipos de diagramas UML, características y propiedades de manera que el aprendiz los utilice dependiendo de las necesidades del sistema de información. Se proporciona inmerso en el contenido la aplicación de un ejemplo de modelado UML como caso práctico para afianzar conocimientos.

Mapa de contenido



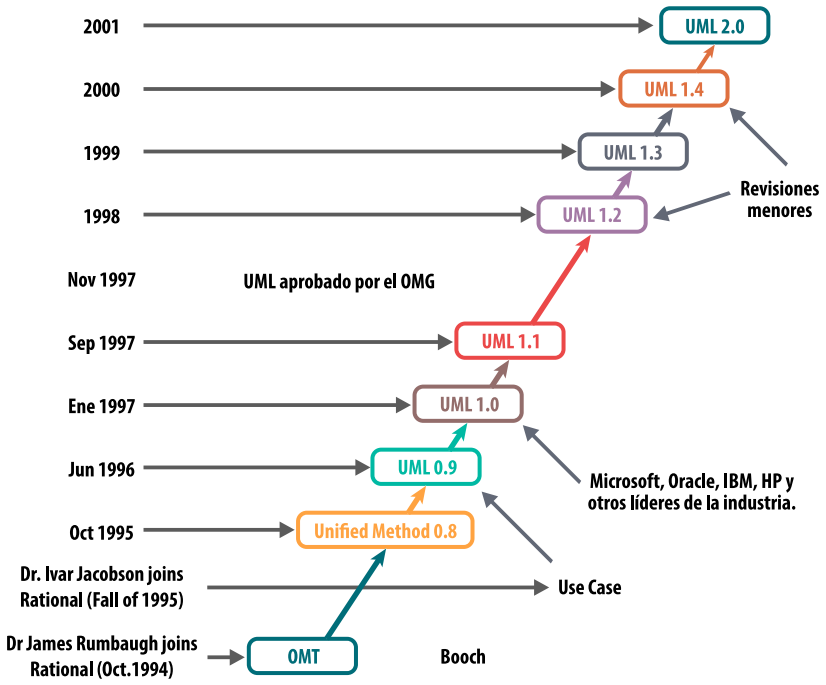
Desarrollo de contenidos

1. El lenguaje unificado de modelado "UML"

El origen del lenguaje unificado de modelado (UML: "Unified Modeling Language") comenzó a gestarse desde octubre de 1994, cuando Rumbaugh se unió a la compañía Rational fundada por Booch (dos reputados investigadores en el área de metodología del software) y cuyo objetivo era la unificación de los métodos Booch y el OMT (Object Modelling Tool), naciendo la primera versión de UML.

El objetivo era suplir la necesidad de representar de manera gráfica y estandarizada la representación de un modelo, lo cual impedía que los diseños gráficos realizados se pudieran compartir fácilmente entre distintos diseñadores. Por lo anterior, se necesitaba un lenguaje no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo, nació el Lenguaje Unificado de Modelado, convirtiéndose en un estándar que permite representar y modelar la información con la que se trabaja en las fases de análisis y diseño.

La versión 1.1 de UML, de noviembre de 1997, fue adoptada por el OMG (Object Management Group http://www.omg.org) como estándar.



Fuente: Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.

Se conoce como **Modelado Visual** cuando se captura las partes esenciales del sistema, se realiza y se plasma en una notación gráfica. El objetivo del modelado visual es permitir manejar la complejidad de los sistemas a analizar o diseñar. Otro objetivo del modelado visual, es la independencia del lenguaje de programación, de tal manera que los diseños realizados usando UML se puedan implementar en cualquier lenguaje orientado a objetos.

1.1 Características de UML

UML es ante todo un lenguaje que se centra en la representación gráfica de un sistema. Este lenguaje indica cómo crear y leer los modelos, representando flujos de trabajo en elementos gráficos.

Dentro de las características del lenguaje de modelado UML:

- **Visualizar:** UML permite modelar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar:** UML permite especificar cuáles son las características de un sistema antes de su desarrollo.
- **Construir:** por medio de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar:** los propios elementos gráficos sirven como documentación del sistema desarrollado, lo cual ayudará al mantenimiento de las soluciones conceptuales en el transcurso del tiempo.

Un modelo UML está compuesto por tres tipos de bloques de construcción:

- **Elementos o partes:** los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, personas, sistemas) que pueden relacionarse. Tienen identidad y se distinguen entre ellos.
- **Relaciones o acciones:** relacionan los elementos entre sí, hacen que los sistemas tengan funcionalidad, que adquieran vida.
- **Diagramas o diseños:** reflejan gráficamente el comportamiento y las relaciones entre los elementos.

Todo sistema puede ser representado gráficamente mediante modelos empleando UML el cual soporta sus diseños usando diagramas que facilitan el entendimiento, trazabilidad y comprensión entre sus componentes brindando claridad y calidad en la información presentada a los diversos actores involucrados en la distribución de información que requiere ser procesada por un sistema o flujo de trabajo propio de una organización empresarial.

2. Tipos de diagramas UML

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones, ofreciendo una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece varios tipos de diagramas que permiten visualizar un sistema desde varias perspectivas. UML incluye los siguientes diagramas:

Diagrama de clases

Es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos.
El diagrama de clases muestra un conjunto de clases, atributos, operaciones, interfaces y sus relaciones.

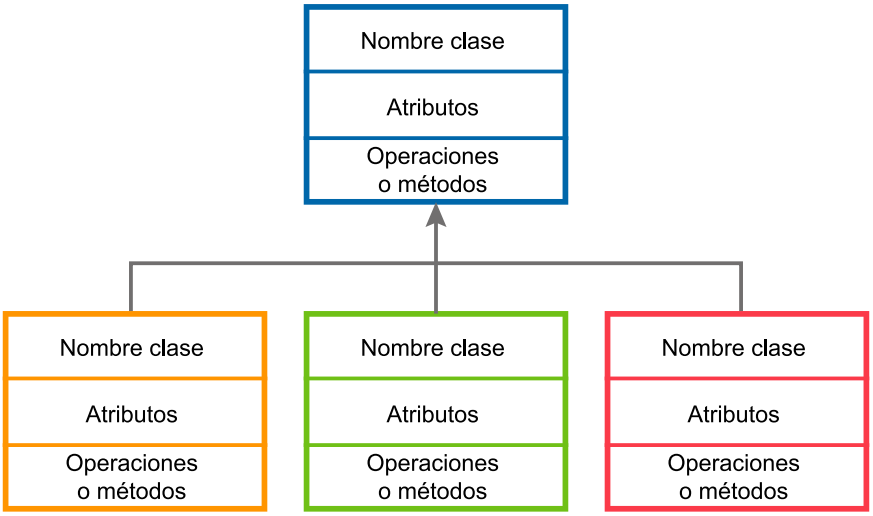


Diagrama de actividades

El diagrama de actividades permite mostrar la secuencia de las acciones de varios objetos y seleccionar el orden en que se harán las cosas.
Representa los procesos de negocios de alto nivel, incluidos el flujo de datos. También puede utilizarse para modelar lógica compleja y/o paralela dentro de un sistema.
Un diagrama de actividades es la versión UML de un diagrama de flujo y se usan para analizar procesos.

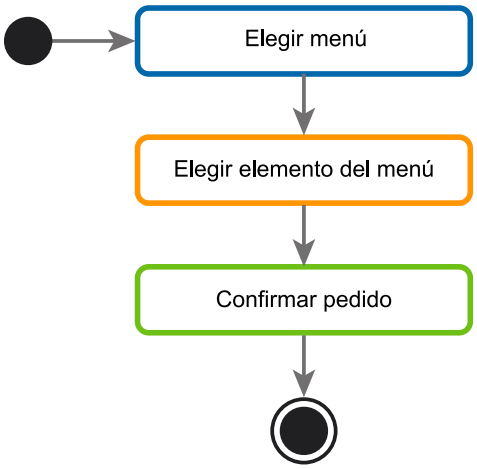


Diagrama de secuencias

En el diagrama de secuencia se muestran las clases a lo largo de la parte superior y los mensajes enviados entre esas clases, modelando un solo flujo a través de los objetos del sistema.
Un diagrama que representa una interacción, poniendo el foco en la secuencia de los mensajes que se intercambian, junto con sus correspondientes ocurrencias de eventos en las líneas de vida.

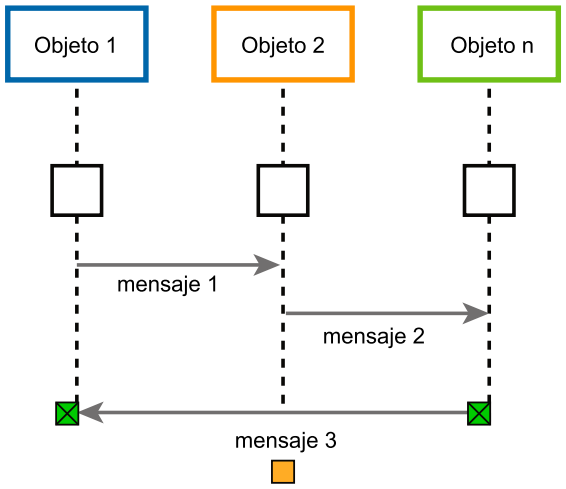


Diagrama de casos de uso

Un diagrama que muestra las relaciones entre los actores y el sujeto (sistema), y los casos de uso.
Modela la funcionalidad del sistema según la percepción del usuario externo (actor) y las transacciones entre los actores y el sistema.
Los diagramas de casos de uso son responsables principalmente de documentar los macrorrequisitos del sistema. Piense en los diagramas de casos de uso como la lista de las funcionalidades que debe proporcionar el sistema.

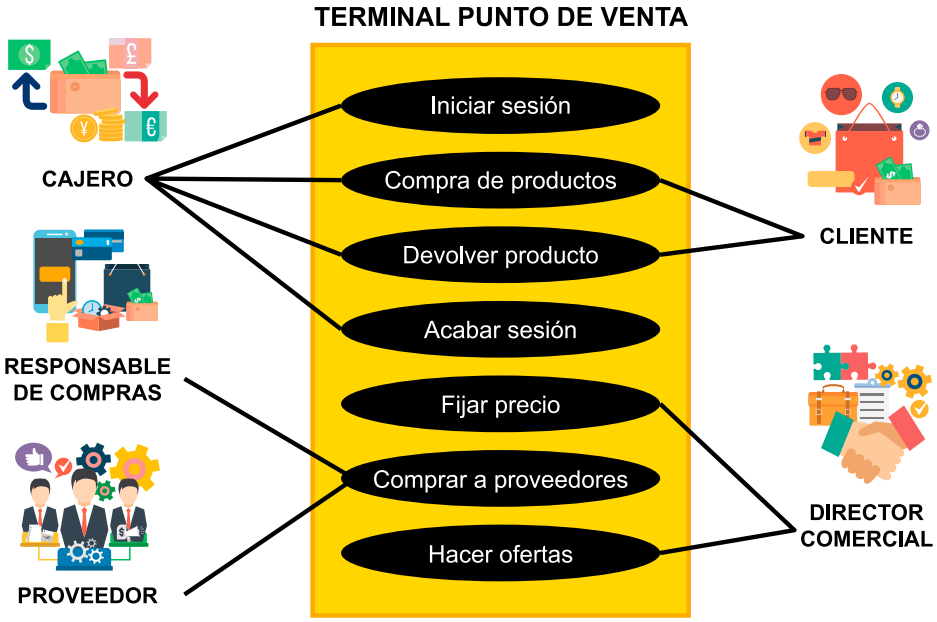
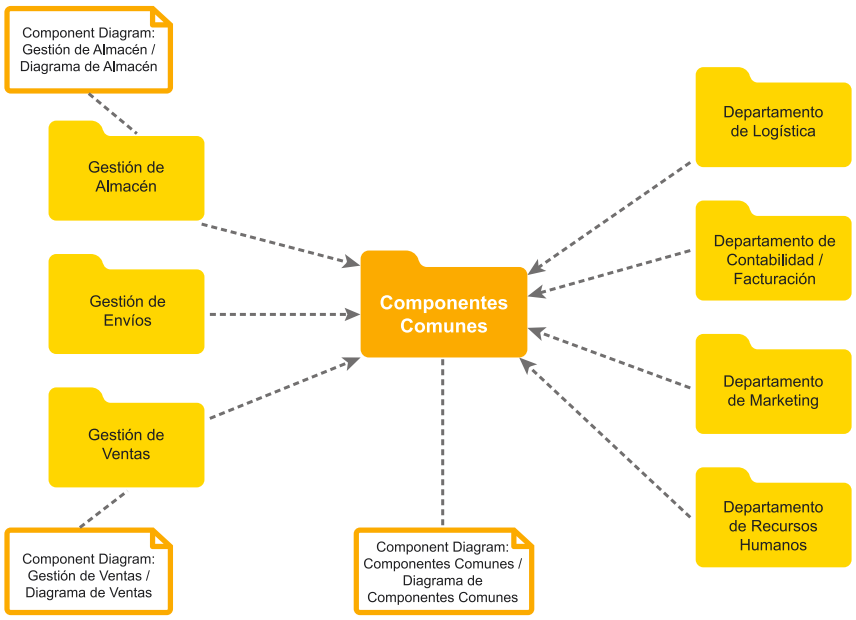


Diagrama de componentes

Representa los componentes que conforman una aplicación, sistema o empresa. Los componentes, sus relaciones, interacciones y sus interfaces públicas.
En esta vista se modelan los componentes del sistema, a partir del cual se construye la aplicación. Son importantes para construir sistemas más grandes a partir de partes pequeñas.



Otros tipos de diagramas:

Diagrama	Descripción
Diagrama de estado	Un diagrama de estados muestra el estado cambiante o dinámica de un objeto. Son importantes en el modelado de comportamiento de una interfaz, clase ante los eventos disparadores del objeto. Cada estado modela un periodo de tiempo, durante la vida de un objeto. Cuando ocurre un evento, se puede desencadenar una transición que lleve el objeto a un nuevo estado.

Diagrama de despliegue físico	Un diagrama de despliegue físico muestra cómo y dónde se desplegará el sistema. Las máquinas físicas y los procesadores se representan como nodos y la construcción interna puede ser representada por nodos o artefactos embebidos. Como los artefactos se ubican en los nodos para modelar el despliegue del sistema, la ubicación es guiada por el uso de las especificaciones de despliegue. En este tipo de diagrama se muestran los tipos de nodos del sistema y los tipos de componentes que contienen. Aquí el nodo se visualiza como un cubo.
Diagrama de objetos	Un diagrama que presenta los objetos y sus relaciones en un punto en el tiempo. Un diagrama de objetos se puede considerar como un caso especial de un diagrama de clases o un diagrama de comunicaciones.
Diagrama de estructura de composición	Representa la estructura interna de un clasificador (tal como una clase, un componente o un caso de uso), incluyendo los puntos de interacción de clasificador con otras partes del sistema. Un uso adicional que se puede dar a los diagramas de estructura compuesta es para mostrar las partes que colaboran, por ejemplo, en un caso de uso.
Diagrama de paquetes	Diagrama que permite la organización de elementos de modelado en paquetes y las dependencias entre ellos. Los usos más comunes para los diagramas de paquete son para organizar diagramas de casos de uso y diagramas de clase, a pesar de que el uso de los diagramas de paquete no es limitado a estos elementos UML.
Diagrama de comunicaciones	Diagrama que describen los mensajes que transmiten los objetos y muestran las asociaciones que existen entre las clases. Modela las interacciones entre objetos en términos de mensajes en secuencia.

Diagrama de tiempos	Diagrama utilizado para mostrar el cambio en el estado o valor de uno o más objetos en el tiempo o en su línea de vida, en respuesta a los eventos o estímulos recibidos. Los eventos que se reciben se anotan, a medida que muestran cuándo se desea mostrar el evento que causa el cambio en la condición o en el estado.



A continuación, se presenta una situación que representa la gestión de información en un centro médico que permitirá una mejor interpretación de los diferentes tipos de diagramas.

3. Caso de estudio práctico

El propietario de un centro médico de su ciudad, requiere con urgencia la construcción de un sistema de información que le permita administrar los datos básicos de sus pacientes, tratamientos, citas y gestión de reportes de tal manera que al ejemplificar el sistema se pueda fijar claramente el límite de este.

Diagramas iniciales: a continuación, se relacionan los diagramas iniciales, utilizados en la fase de análisis de un sistema de información.

3.1 Casos de uso

Al momento de desarrollar un proyecto se debe pensar en cuáles serán las principales funcionalidades que el software debe permitir llevar a cabo y quiénes serán los que podrán ejecutar dichas funcionalidades. La identificación de estos elementos se puede visualizar de manera efectiva a través de la elaboración de diagramas de Casos de Uso. Estos diagramas que son elaborados durante las etapas iniciales de un proyecto se convierten en un referente para cada una de las etapas siguientes del desarrollo del proyecto.

3.1.1 Componentes

Sistema: se representa mediante un rectángulo. Este se emplea para delimitar gráficamente lo que se encuentra por dentro del sistema (los casos de uso) y lo que está por fuera del sistema (los actores).



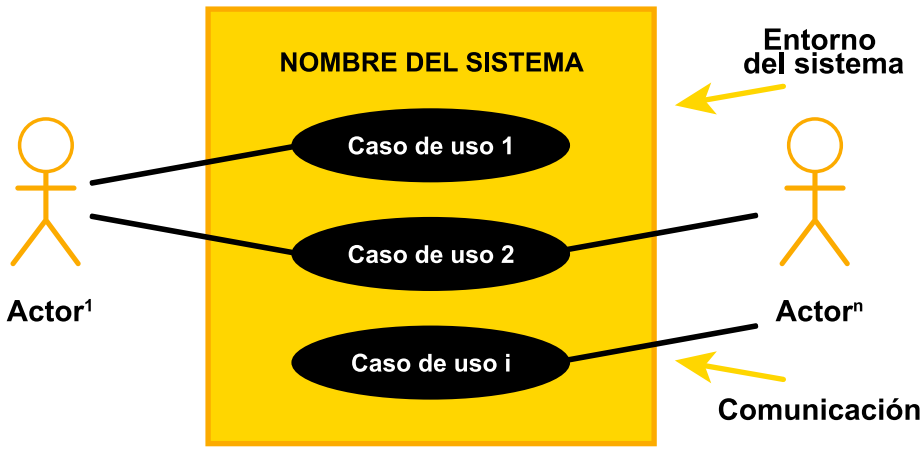
Actor: se representa mediante un "hombre de palo". Este se emplea para indicar el tipo de usuario del sistema que podrá ejecutar alguna función (caso de uso) en el mismo.



Caso de uso: se representa mediante un óvalo e indica una función que el sistema debe realizar.



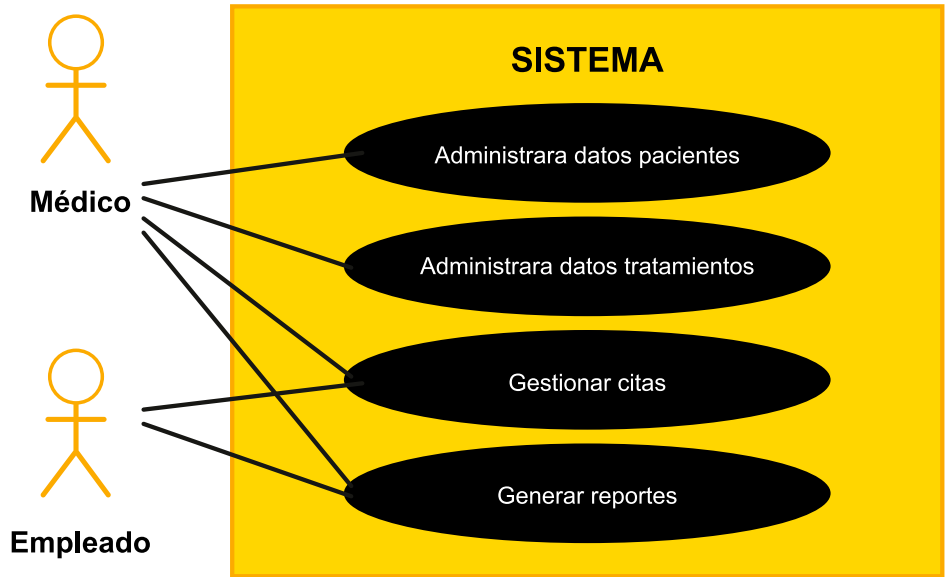
Ejemplo de la representación gráfica de un Diagrama de Caso de Uso:



Para nombrar un proceso se puede emplear un verbo conjugado en infinitivo y que represente la función a realizar (Administrar, Gestionar, Registrar, entre otros). Por ejemplo:

- Administrar los datos de pacientes.
- Gestionar citas médicas.
- Administrar datos de historias clínicas o tratamientos.
- Generar reportes.

3.1.2 Representación gráfica



Identificación de casos de uso: en el ejemplo anterior se observan los casos de uso identificados en el sistema, es decir, las funcionalidades que el sistema va a proveer (Administrar datos pacientes, Administrar datos tratamientos, Gestionar citas, Generar reportes).

Identificación de actores: los actores son los usuarios externos que podrán ejecutar los casos de uso, en el ejemplo anterior, se identificaron dos actores (Médico y Empleado).

Las líneas que van del actor al caso de uso se denominan Asociación y sirven para determinar cuáles Casos de uso lleva a cabo un determinado Actor.

3.1.3 Tipos de relaciones

Asociación: es la relación que se da entre los actores y el caso de uso. Esta relación indica que el actor lleva a cabo el caso de uso.



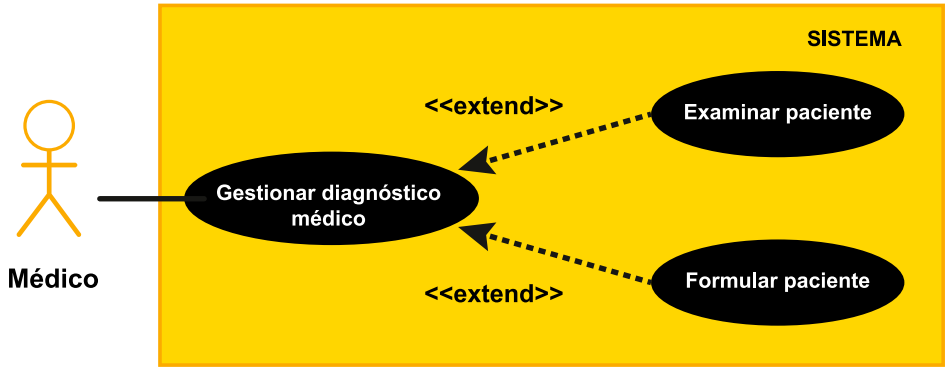
El ejemplo anterior indica que el Médico dentro del sistema Sismédico puede llevar a cabo la funcionalidad de Administrar Datos Pacientes.

Incluido ó <<uses>> ó <<include>>: simboliza la relación que se da entre casos de uso donde uno termina incluyendo al otro (describe el comportamiento de uno en otro).



El ejemplo anterior indica que para Gestionar una cita se debe validar datos, es decir, el caso de uso Gestionar Cita “usa” al caso de uso Validar Datos.

Extendido ó <<extend>>: relación que se da entre casos de uso donde se evidencia que un caso de uso es la generalización o especialización de otro.



El ejemplo anterior indica que, al gestionar un diagnóstico médico, se puede examinar un paciente o formular un paciente, es decir, los casos de uso FORMULAR A PACIENTE y EXAMINAR A PACIENTE se extienden del caso de uso GESTIONAR DIAGNÓSTICO MÉDICO.

3.1.4 Documentación

La técnica de casos de uso requiere además de construir el diagrama de Casos de Uso, la descripción de los mismos. Esta descripción permite detallar el flujo de eventos que se da entre el Sistema y el Actor para llevar a cabo el Caso de Uso. A continuación, se presenta el formato que describe el Caso de Uso del centro médico.

CASO DE USO	ADMINISTRAR DATOS PACIENTES	
Descripción	El comportamiento del sistema deberá describir el paso a paso del caso de uso cuando el personal encargado de gestionar datos del paciente inicie el ingreso de estos.	
Precondición	El paciente no se encuentra ingresado al sistema y tiene la documentación necesaria para poder ser ingresado al sistema.	
Secuencia Normal	Paso	Acción
	1	El personal médico ingresa al sistema para registrar el nuevo paciente.
	2	El sistema carga formulario para registro de datos del paciente así: identificación, nombre(s), apellido(s), dirección, teléfono, estrato, tipo de RH, género, acudiente.
	3	El personal médico ingresa los datos suministrados por el paciente y ejecuta la acción en el sistema.
	4	El sistema almacena los datos suministrados por el personal médico, imprime el carnet del Sistema General de Seguridad Social en Salud, el sistema comunica al personal médico que el proceso ha terminado de manera exitosa.
	5	El personal médico genera reporte del sistema de registro al nuevo paciente, mediante la expedición del carnet o constancia de inscripción al sistema de Seguridad Social en Salud.
PostCondición	El paciente se encuentra registrado en el Sistema General de Seguridad Social en Salud, su historial clínico es nuevo.	
Excepciones	Paso	Acción
	1	Si el sistema detecta la duplicación de un paciente registrado con la identificación que se registra, procede a informar al personal médico, estos deben modificar y/o actualizar la información que sea necesaria y continuar el caso de uso.
	2	Si el personal médico cancela el registro del paciente se termina el caso de uso.

3.2 Diagrama de secuencia

Los diagramas de secuencia se utilizan para describir el funcionamiento interno del sistema desde el punto de vista de la implementación y son parte fundamental de la vista de interacción de los diagramas UML.

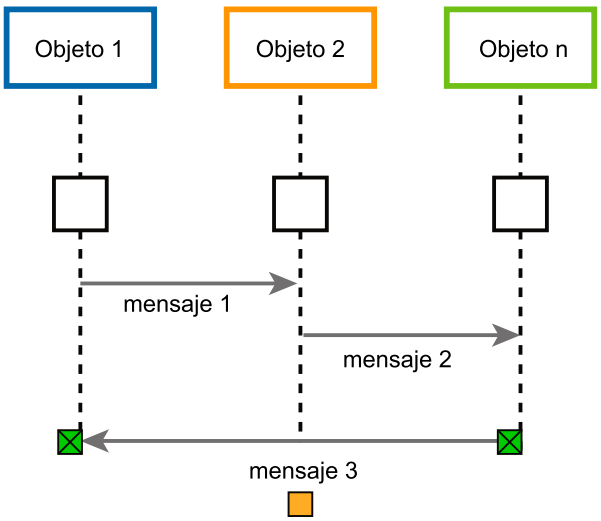
En un diagrama de secuencia, un objeto se muestra como caja en la parte superior de una línea vertical punteada. Esta línea vertical se llama línea de vida del objeto. La línea

de vida representa la vida del objeto durante la interacción. Cada mensaje se representa mediante una flecha entre las líneas de vida de dos objetos. El orden en el que se dan estos mensajes transcurre de arriba hacia abajo.

Una de las cuestiones difíciles de comprender en un programa orientado a objetos "POO", es el flujo de control general. Un buen diseño puede tener pequeños métodos en diferentes clases, y a veces resulta muy complicado determinar la secuencia global de comportamiento.
Los diagramas de secuencia le ayudan a ver la secuencia.

3.2.1 Representación gráfica

Mientras que el diagrama de casos de uso permite el modelado de una vista del escenario, un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales.



Para la construcción de un diagrama de secuencia tener presente:

- Dibujar una línea vertical que representa el sistema.
- Dibujar una línea para cada actor que interactúa directamente con el sistema.
- Del curso de eventos del caso de uso, identificar los eventos externos generados por los actores. Mostrarlos en el diagrama.

3.2.2 Tipos de mensajes

Existen dos tipos de mensajes: sincrónicos y asincrónicos.

- **Mensajes sincrónicos:** se corresponden con llamadas a métodos del objeto que recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensajes se representan con flechas con la cabeza llena.

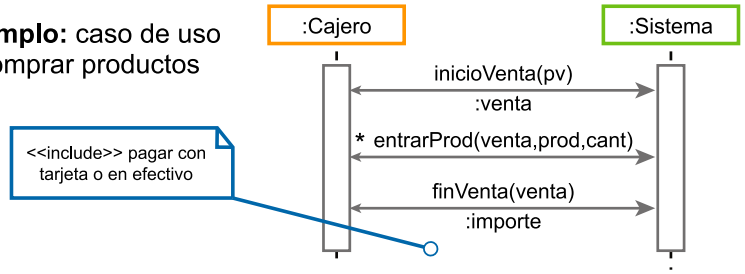
- **Mensajes asincrónicos:** los mensajes asincrónicos terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia. Se representan con flechas con la cabeza abierta. También se representa la respuesta a un mensaje con una flecha discontinua.

Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria. Durante el análisis inicial, el modelador típicamente coloca el nombre 'business' de un mensaje en la línea del mensaje. Más tarde, durante el diseño, el nombre 'business' es reemplazado con el nombre del método que está siendo llamado por un objeto en el otro. El método llamado, o invocado, pertenece a la definición de la clase instanciada por el objeto en la recepción final del mensaje.

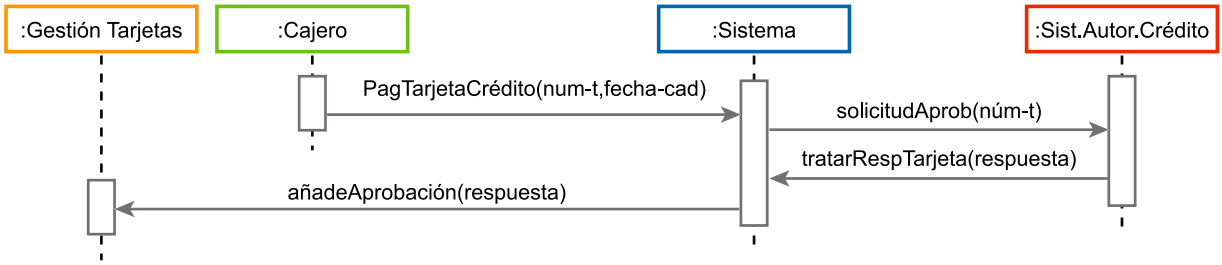
3.2.3 Diagrama de secuencia <<include>>

Los casos de uso definidos mediante <<include>> requieren un diagrama de secuencia para la parte común y uno para cada caso de uso que es incluido.

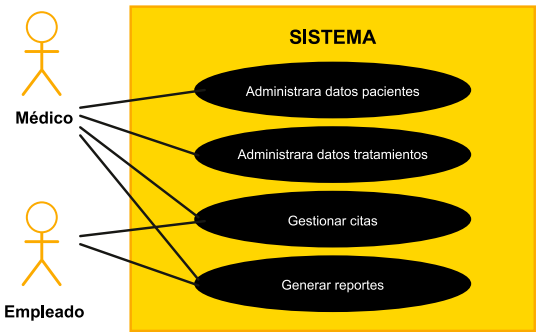
Ejemplo: caso de uso comprar productos



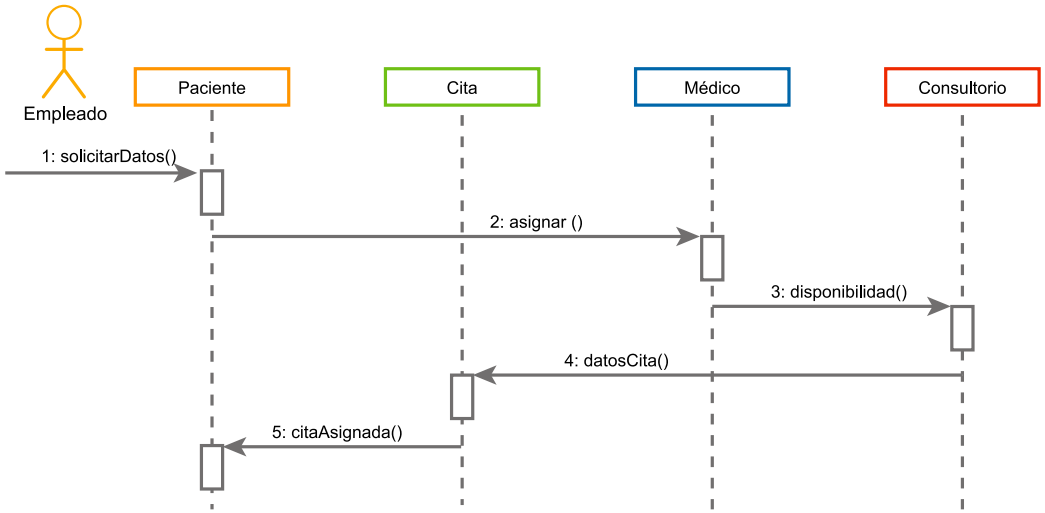
Parte específica: pagar con tarjeta.



Volviendo al ejemplo del caso de uso "Gestionar Citas", cuyo diagrama es:



El respectivo diagrama de secuencia correspondiente al caso de uso “Gestionar Citas” es el siguiente:



3.3 Diagrama de comunicación (Colaboración)

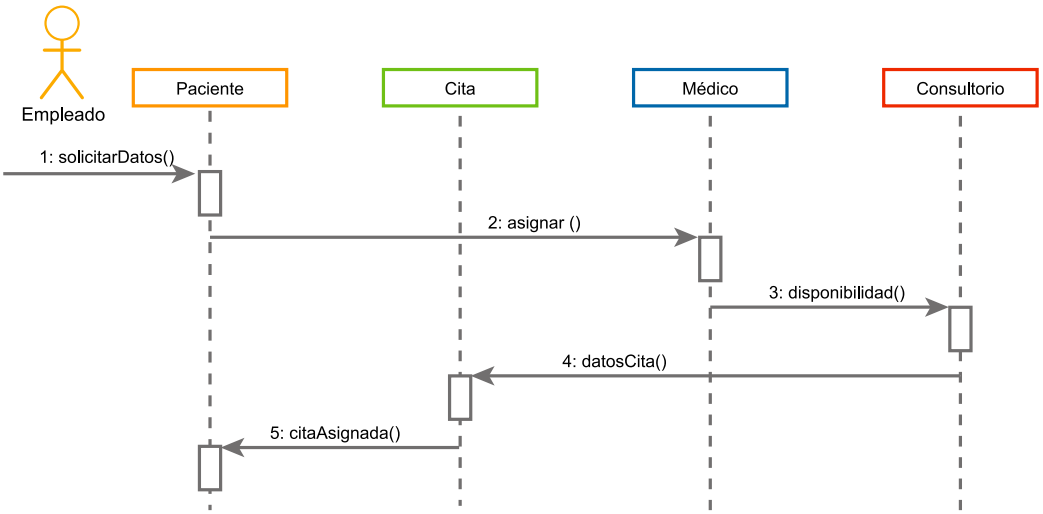
Un Diagrama de Colaboración (versión UML 1.0) es luego llamado Diagrama de Comunicación (versión UML 2.0), transmite la misma información que un diagrama de secuencia. La diferencia está en que el diagrama de Secuencia hace énfasis en el orden y secuencia en que se emiten mensajes entre los objetos para cumplir con un determinado proceso e incluye dentro de sus componentes la línea de vida para cada objeto, mientras que el diagrama de Comunicación hace énfasis en la relación entre los objetos para satisfacer una operación, generando una visión espacial del sistema en la ejecución de un proceso.

El Diagrama de Secuencia, hace énfasis en la secuencia; es fácil apreciar el orden en el que ocurren las cosas, mientras que el Diagrama de Comunicación, usa la distribución para indicar cómo se conectan estáticamente los objetos.

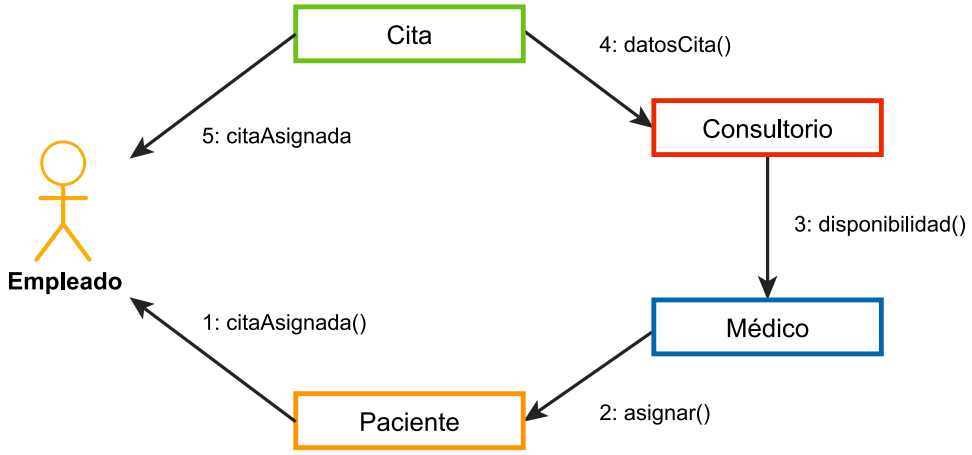
3.3.1 Representación gráfica

Los componentes gráficos del diagrama de Comunicación son los mismos del diagrama de Secuencia excepto por el no uso de la línea de vida de los objetos y la necesidad de incluir un número en los mensajes que identifique el orden en que se deben ejecutar.

La numeración de los mensajes determinará el orden en que estos se ejecutan.
Para ilustrar el uso de diagramas de Comunicación, se toma el ejemplo del diagrama de Secuencia que se ha venido trabajando como caso de estudio.



El correspondiente diagrama de comunicación sería:



3.4 Diagrama de actividades

Diagrama de actividades se centra en mostrar el flujo de actividades dentro de un sistema. Los diagramas de actividades cubren la parte dinámica de un sistema y se utilizan para modelar el funcionamiento de un sistema resaltando el flujo de control entre objetos.
Los diagramas de actividades tienen una serie de beneficios para toda organización. Entre los beneficios están:

- Ilustrar un proceso de negocios o flujo de trabajo entre los usuarios y el sistema.
- Demostrar la lógica de un algoritmo.

- Describir los pasos realizados en un caso de uso UML.
- Simplificar y mejorar cualquier proceso clarificando casos de uso complicados.
- Modelar elementos de arquitectura de software, tales como método, función y operación.

Este diagrama es similar al Diagrama de Flujo que se emplea para representar gráficamente un algoritmo, teniendo como principal diferencia que el diagrama de actividades permite la ejecución de actividades (pasos) en paralelo.
Dentro del mundo de la orientación a objetos y el UML, un diagrama de actividades es usado comúnmente para representar los pasos lógicos requeridos para llevar a cabo un Caso de Uso.

3.4.1 Representación gráfica

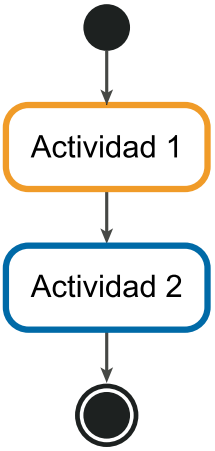
Inicio: todo diagrama de actividades debe poseer un único inicio, el cual se representa mediante un círculo relleno.



Fin: indica que el proceso ha terminado. Se representa mediante un círculo que rodea al círculo relleno.

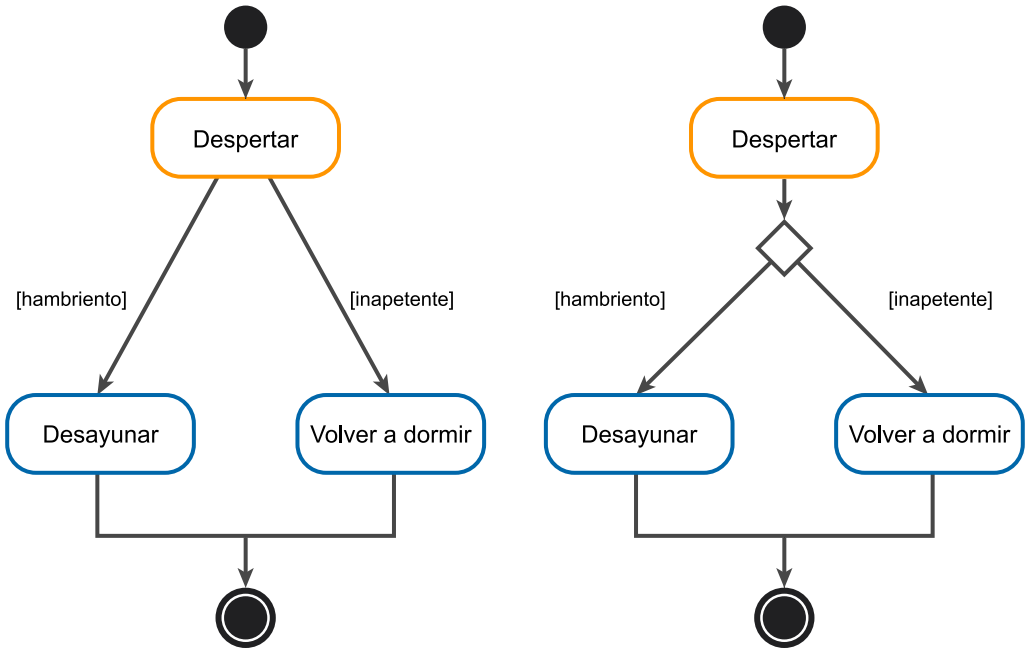


Actividades: indica una acción requerida para llevar a cabo un proceso. Se representa mediante un rectángulo con sus vértices redondeados. Las flechas que van dirigidas de una actividad a otra se denominan transiciones.

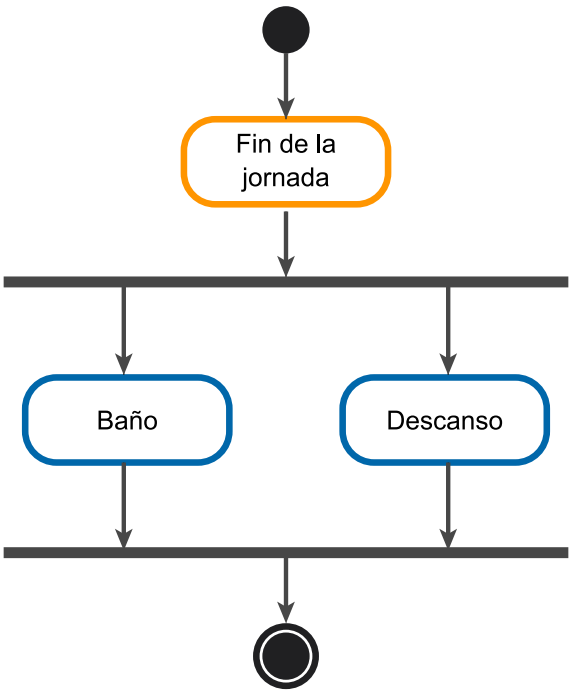


Decisión: tal como sucede con los diagramas de flujo, en el diagrama de actividades se puede llegar a un punto donde de acuerdo con el cumplimiento o no de una condición se debe tomar por uno u otro camino.

Existen dos formas de representar la decisión en un diagrama de actividades, partiendo libremente de la actividad hacia alguno de los dos caminos o llegando a un rombo desde donde se parte hacia cualquiera de los dos caminos.

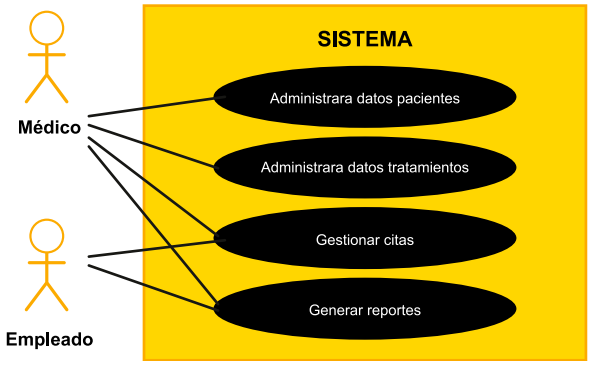


Rutas concurrentes: la diferencia clave entre un diagrama de actividades y un diagrama de flujo es que los diagramas de flujo se limitan normalmente a procesos secuenciales; los diagramas de actividades pueden manejar procesos paralelos, es decir, actividades concurrentes que se pueden llevar a cabo en el mismo momento y después se unen a otra actividad. El inicio como el fin de una ruta concurrente se representa mediante una línea horizontal sólida.

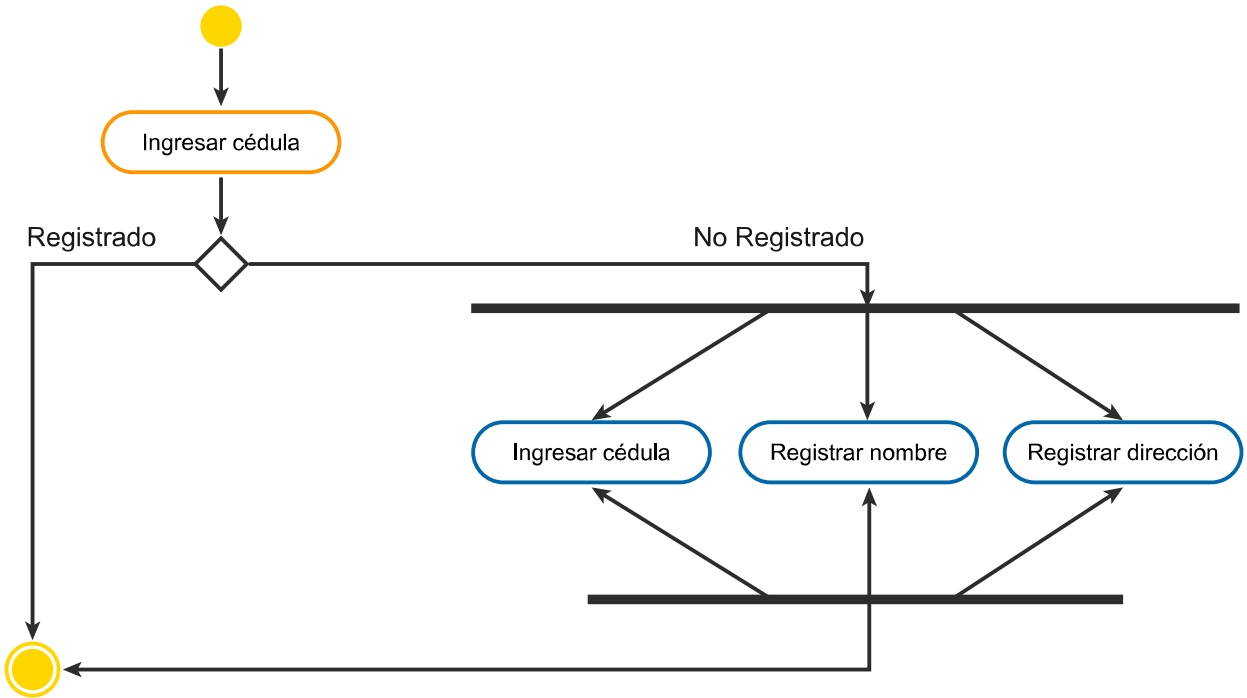


Indicaciones: durante la secuencia de actividades es posible enviar una indicación para que al recibirla, se pueda ejecutar una actividad. Se representa mediante pentágonos que se acoplan generando la sensación de envío y recepción de la indicación.

Volviendo al ejemplo del caso de uso “Gestionar Citas”, cuyo diagrama es:



El diagrama de Actividades correspondiente para el caso de uso Administrar Datos Pacientes es:



3.5 Diagrama de clases

Los diagramas de clases se usan para mostrar las clases de un sistema y las relaciones entre ellas. Una clase es un elemento importante dentro del contexto de un sistema, que puede tener información o datos valiosos y realizar acciones que sean necesarias dentro del funcionamiento del sistema.

Por ejemplo, en un software para un supermercado, seguramente los elementos más importantes sobre los cuales sea importante mantener información son los productos, los clientes, las ventas y los pedidos, en este caso se han encontrado las clases PRODUCTO, CLIENTE, VENTA y PEDIDO.

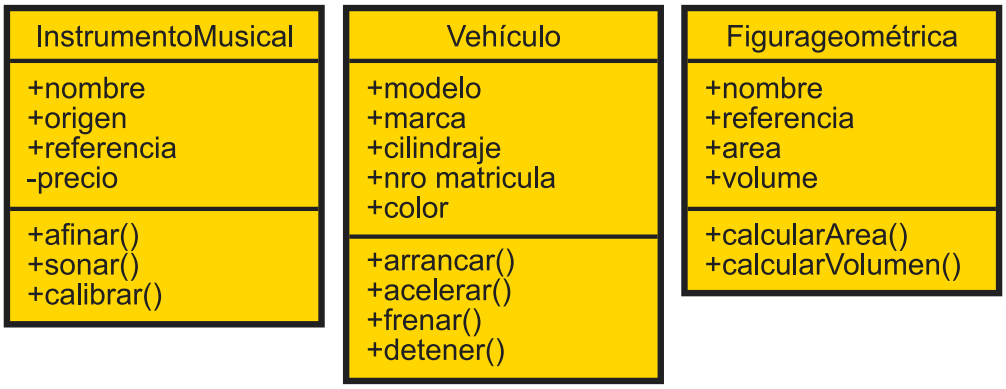
Estas clases a su vez tienen atributos (datos) y métodos (funciones), por ejemplo, la clase PRODUCTO tiene como uno de sus atributos el atributo precio y uno de sus métodos puede ser incrementarPrecio. De esta forma, a través de los atributos se puede acceder a la información de la clase y a través de los métodos se pueden ejecutar acciones sobre la clase. Estas clases se unen a otras clases a través de relaciones y así se conforma el diagrama de clases.

3.5.1 Clases

Una clase es la unidad básica que agrupa una colección de objetos que poseen un tipo de comportamiento. Toda clases se compone de los siguientes elementos:

- Nombre de la clase.
- Atributos, datos o propiedades también denominados miembros de la clase.
- Métodos (operaciones) o acciones propias de la clase. Estas acciones se identifican con verbos en infinitivo.

Ejemplos de clases:



ATRIBUTOS: los atributos o características de una Clase pueden ser de tres tipos, que definen su visibilidad:

- **Public (+):** indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados. Disponible para consumo externo.
- **Private(-):** indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden acceder). Disponible para consumo interno.
- **Protected (#):** indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven (ver herencia). Disponible para consumo interno o consumo por parte de las clases hijos.
- **Sin modificador de acceso ( ):** indica que el atributo será accesible desde cualquier clase que se encuentre en el mismo paquete de la clase que contiene al atributo sin modificador de acceso.

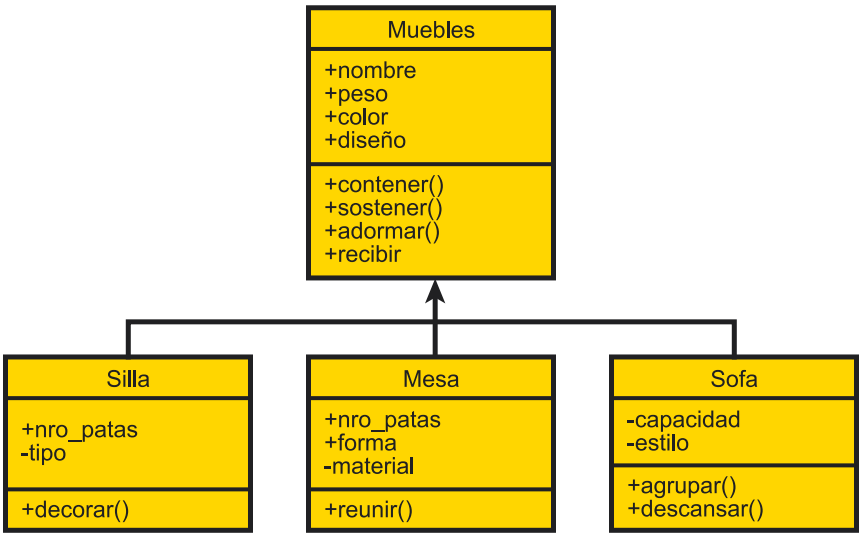
MÉTODOS: los métodos u operaciones de una clase son la forma en cómo ésta interactúa con su entorno, éstos pueden tener las características.

- **Public (+):** indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados. Disponible para consumo externo.
- **Private (-):** indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden acceder). Disponible para consumo interno.
- **Protected (#):** indica que el método no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven (ver herencia). Disponible para consumo interno o consumo por parte de las clases hijos.
- **Sin modificador de acceso ( ):** indica que el método será accesible desde cualquier clase que se encuentre en el mismo paquete de la clase que contiene al método sin modificador de acceso.

3.5.2 Relaciones

Las relaciones entre las clases se dan por Herencia, Asociación, Agregación, Composición y Dependencia (uso).

Herencia (Especialización / Generalización)



En el ejemplo anterior se ilustra que un Mueble puede ser una Silla, una Mesa o un Sofá.

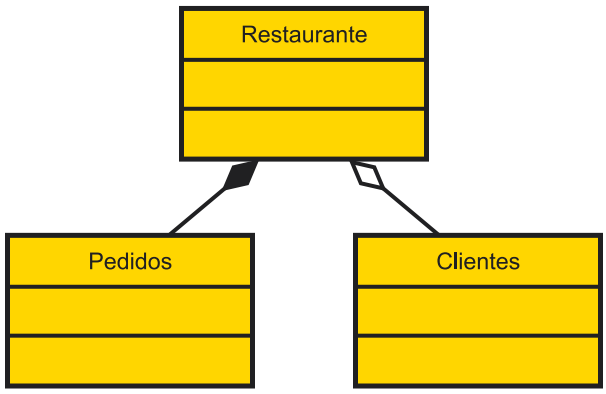
Agregación y composición

- **Composición por valor:** relación estática, donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada Composición (el Objeto base se construye a partir del objeto incluido, es decir, es “parte/todo”).

  Se representa de forma gráfica: 

- **Agregación por referencia:** tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada Agregación (el objeto base utiliza al incluido para su funcionamiento).

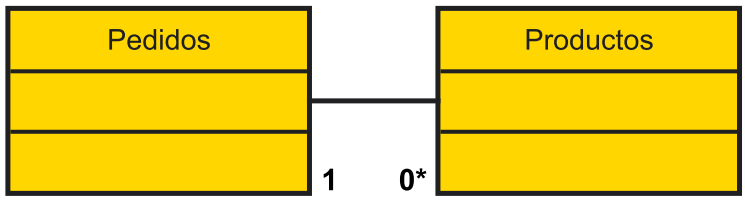
  Se representa de forma gráfica: 



En el ejemplo anterior indica que el Restaurante tiene Pedidos y Clientes, sin embargo, los Pedidos requieren del Restaurante para poder existir (Composición), mientras que los Clientes no (Agregación).

Asociación

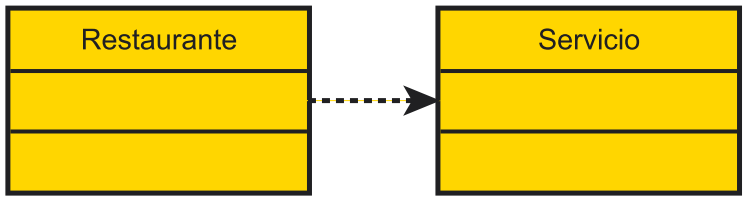
La relación entre clases, permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.



El ejemplo anterior indica que un Pedido está relacionado con cero o muchos Productos.

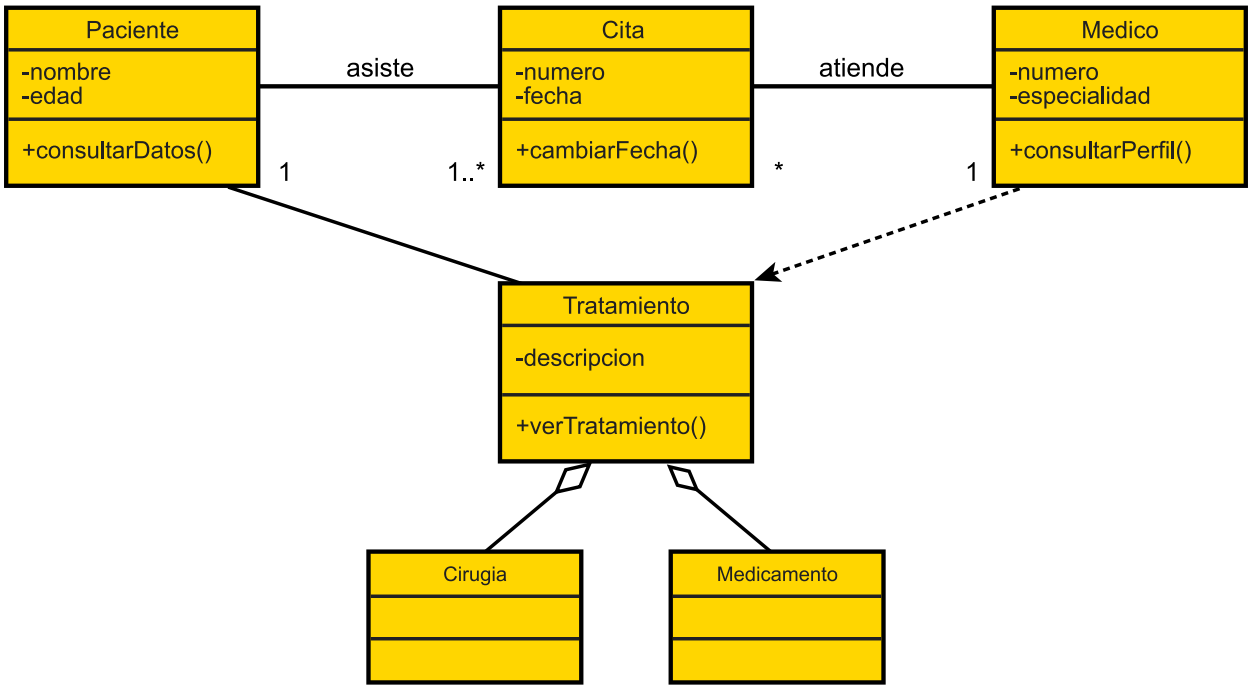
Dependencia

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).



En el ejemplo anterior se indica que la clase Restaurante instancia o crea objetos de la clase Servicio.

Volviendo al ejemplo del caso de uso “Gestionar Citas”, el respectivo diagrama de clases es:



3.6 Diagrama de objetos

Un diagrama de objetos es en esencia muy similar a un diagrama de clases, su propósito es modelar el sistema en un momento determinado (un punto en el tiempo) a partir de las posibles instancias que se deriven de las clases.

Este diagrama se usa para hacer más entendible el sistema modelado ya que en lugar de presentar únicamente el nombre de la clase como algo general, se presenta el posible objeto derivado de la clase, dando mayor claridad y especificación al sistema.

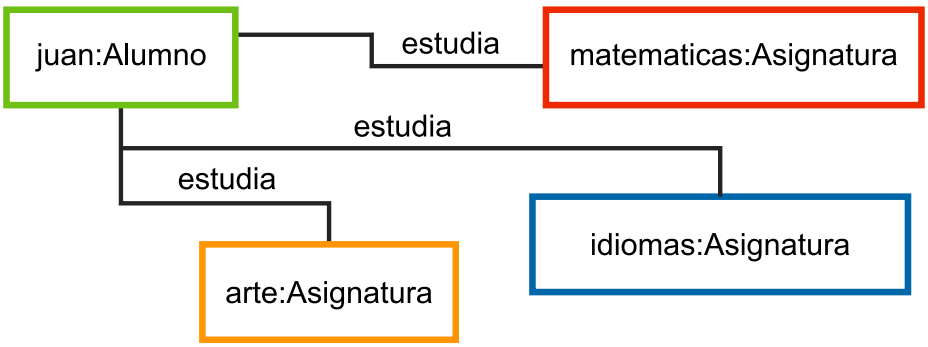
3.6.1 Representación gráfica

En el diagrama de Objetos se emplean los mismos componentes que en el diagrama de Clases pero se omite la multiplicidad, la cual se puede observar de manera explícita cuando se incorpora una o varias instancias de una clase en el diagrama.

Sintaxis de los nombres de los objetos:

NombreObjeto:NombreClase, de esta manera, para representar un objeto de la clase Asignatura, su nombre sería por ejemplo: matemáticas:Asignatura

El siguiente ejemplo ilustra una asociación presente en un diagrama de Objetos, a través de la cual se visualiza que un alumno estudia diferentes asignaturas en una instancia completa y específica:



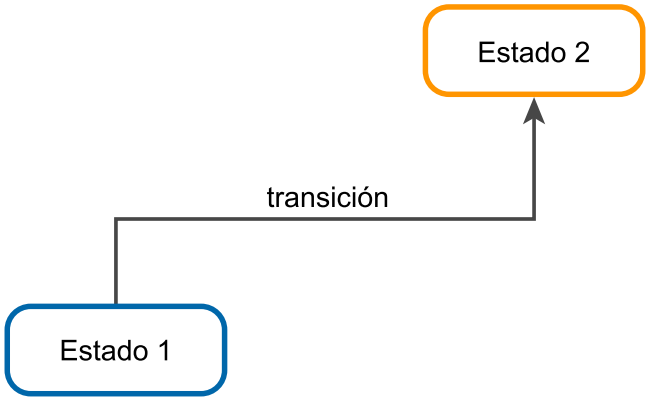
3.7 Diagrama de estados

Un diagrama de estados, muestra la secuencia de estados en los que pasa un objeto durante su vida, en respuesta a los eventos o estímulos recibidos, junto con sus respuestas. Igualmente, muestra los eventos que implican el cambio de un estado a otro.

Los diagramas de objetos usan un subconjunto de elementos de un diagrama de clase para hacer énfasis en la relación entre las instancias de las clases en algún punto en el tiempo. Estos son útiles para entender los diagramas de clases. Estos no muestran nada diferente en su arquitectura a los diagramas de secuencia, pero reflejan multiplicidad y roles.

3.7.1 Representación gráfica

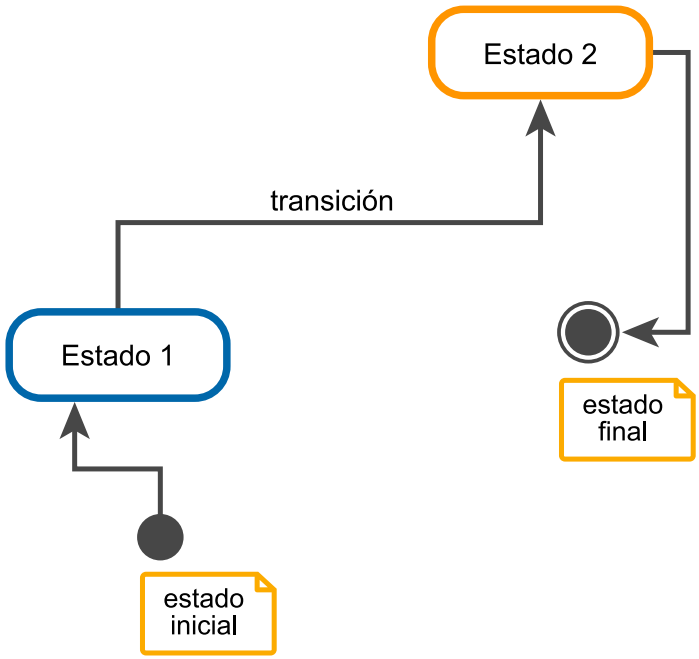
El estado de un componente o sistema representa algún comportamiento que es observable externamente y que perdura durante un periodo de tiempo finito. Viene dado por el valor de uno o varios atributos que lo caracterizan en un momento dado.



Una transición es un cambio de estado producido por un evento y refleja los posibles caminos para llegar a un estado final desde un estado inicial. Desde un estado pueden surgir varias transiciones en función del evento que desencadena el cambio de estado.

Características:

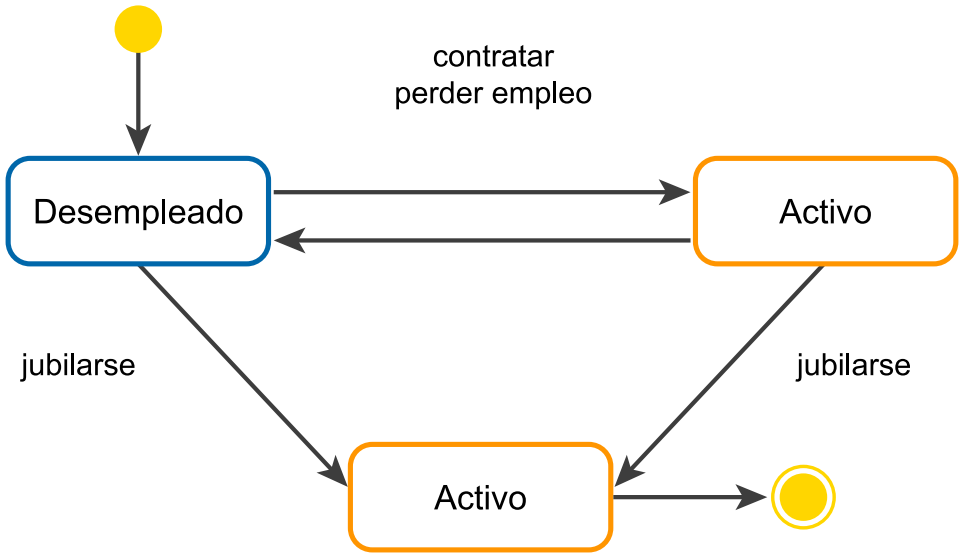
- Un sistema sólo puede tener un estado inicial, que se representa mediante una transición sin etiquetar al primer estado normal del diagrama.
- Pueden existir varias transiciones desde el estado inicial, pero deben tener asociadas condiciones, de manera que sólo una de ellas sea la responsable de iniciar el flujo.
- En ningún caso puede haber una transición dirigida al estado inicial.
- El estado final representa que un componente ha dejado de tener cualquier interacción o actividad.
- No se permiten transiciones que partan del estado final.
- Puede haber varios estados finales en un diagrama, ya que es posible concluir el ciclo de vida de un componente desde distintos estados y mediante diferentes eventos, pero dichos estados son mutuamente excluyentes, es decir, sólo uno de ellos puede ocurrir durante una ejecución del sistema.



Los diagramas de transición de estados comprenden además otros dos elementos que ayudan a clarificar el significado de los distintos estados por los que pasa un componente o sistema. Estos elementos se conocen como acciones y actividades. Una acción es una operación instantánea asociada a un evento, cuya duración se considera no significativa y que se puede ejecutar dentro de un estado, al entrar en un estado o al salir del mismo.

Una actividad es una operación asociada a un estado que se ejecuta durante un intervalo de tiempo hasta que se produce el cambio a otro estado. Para aquellos estados que tengan un comportamiento complejo, se puede utilizar un diagrama de transición de estados de más bajo nivel. Estos diagramas se pueden mostrar por separado o bien incluirse en el diagrama de más alto nivel, dentro del contorno del estado que representa. En cualquier caso su contenido formará un contexto independiente del resto con sus propios estados inicial y final.

Ejemplo diagrama de estados para el objeto empleado:



Glosario

Actividades: acción requerida para llevar a cabo un proceso.

Actor: se emplea para indicar el tipo de usuario del sistema que podrá ejecutar alguna función.

Asociación: relación que se da entre elementos de los diagramas, por ejemplo entre los actores y el caso de uso o entre las clases.

Atributos: características de una Clase, son datos específicos que interesa guardar de cada entidad.

Cardinalidad: indica la participación que tiene una entidad en la relación.

Caso de uso: indica una función que el sistema debe proveer.

Clases: unidad básica que agrupa una colección de objetos.

Entidad: elemento del sistema de los cuales interesa almacenar información.

Extend: relación que se da entre casos de uso.

Métodos: operaciones de una clase.

Relaciones: también se conoce como asociaciones, sirven para interconectar las entidades.

Transición: es un cambio de estado producido por un evento.

UML: Unified Modeling Language, es un lenguaje estandarizado que se utiliza para visualizar los elementos de un sistema de software, compuesto por diagramas que representan elementos estáticos y dinámicos del sistema.

Bibliografía

Craig, Larman. (2007). *UML y patrones.* Editorial Prentice Hall.

Martin, F., Scott, K. (1997). *UML gota a gota.* Pearson Educación.

Gutiérrez, C. (2011). *Casos prácticos de UML. Editorial Complutense.* Recuperado de http://site.ebrary.com.bdigital.sena.edu.co/lib/senavirtualsp/reader.action?docID=10536104&ppg=1

Rumbaugh, J., Jacobson, I. (2007). *El lenguaje unificado de modelado: manual de referencia.* Addison-Wesley.

Kimmel, Paul. (2008). *Manual de UML Guía de Aprendizaje.* Editorial McGraw-Hill Professional Publishing. Recuperado de http://site.ebrary.com.bdigital.sena.edu.co/lib/senavirtualsp/detail.action?docID=10433806&p00=manual+uml

Teniente López, Ernest. (2003). *Especificación de sistemas software en UML.* Universidad Politécnica de Catalunya. Recuperado de http://site.ebrary.com.bdigital.sena.edu.co/lib/senavirtualsp/detail.action?docID=11046224&p00=uml

Control del documento

CONSTRUCCIÓN OBJETO DE APRENDIZAJE

INTRODUCCIÓN AL LENGUAJE DE MODELADO UML	
Centro Industrial de Mantenimiento Integral - CIMI Regional Santander	
Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Expertos temáticos:	Andrés Julián Valencia - V1 Edgar Eduardo Vega A. - V2
Diseño multimedia:	Eulises Orduz Amezquita
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de la licencia que el trabajo original.

creative commons

BY NC SA

UML, Copyright © Object Management Group, Inc. Todos los derechos reservados.

Registered trademark