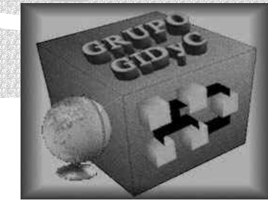
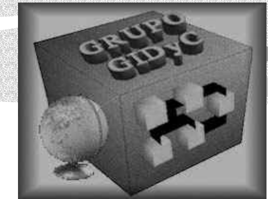
UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA



Introducción al análisis y diseño de algoritmos y la Técnica de Desarrollo de Sistemas de Objetos (TDSO)

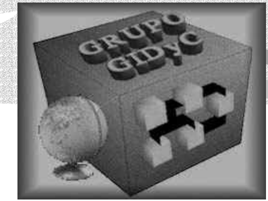
- ✓ INTRODUCCIÓN
- ✓ ORIENTACIÓN POR OBJETOS
- ✓ TIPOS ABSTRACTOS DE DATOS
- ✓ MECANISMOS DE LA ABSTRACCIÓN DE DATOS
- ✓ NOTACIÓN GRÁFICA
- ✓ TDSO
- ✓ CONCLUSIONES



Introducción

- **Algoritmo**: es cualquier secuencia de pasos bien definidos que toma algún conjunto de valores de entrada y produce algún conjunto de valores como salida. (al-Khowârizmi, matemático persa, siglo IX)
- El algoritmo más famoso: algoritmo de Euclides para calcular el máximo común divisor de dos números enteros.
- Un **problema** es una interrogante que debe ser respondida y que normalmente depende de varios parámetros.
- Una **instancia** de un problema es una asignación de valores a sus parámetros
- **Algoritmia:** ciencia que estudia los algoritmos
  - Permite evaluar el efecto de factores externos sobre los algoritmos disponibles
  - Forma de diseñar un nuevo algoritmo para un problema específico

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA



Ejemplos de diversos algoritmos

Multiplicación

```
   1234
 x  981
-------
   1234
  9872
11106
-------
1210554
```

Multiplicación a la inglesa

```
   1234
 x  981
-------
11106
  9872
   1234
-------
1210554
```

Multiplicación a la rusa

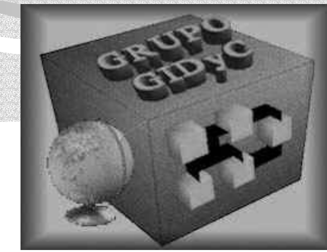
981	1234	1234
490	2468	
245	4936	4936
122	9872	
61	19744	19744
30	39488	
15	78976	78976
7	157952	157952
3	315904	315904
1	631808	631808

Multiplicación usando la técnica divide-y-vencerás

Multiplicar		desplazar	resultado
09	12	4	108….
09	34	2	306..
81	12	2	972..
81	34	0	2754
			1210554

Multiplicar		desplazar	resultado
0	9	2	0..
0	2	1	0.
9	1	1	9.
9	2	0	18
			108

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA



Introducción a TDSO

✓ Antecedentes:

✓Técnicas de diseño: la técnica HIPO [Stay, 1976], el refinamiento paso a paso [Wirth, 1971], el diseño modular y estructurado[Stevens, 1974, Montilva, 1987], los algoritmos estructurados [Montilva, 1982], el método deductivo MEDEE [Dufourd, 1988], etc.

✓ Metodología OMT (Object- oriented modeling technique) [Rumbaugh, 1991], que describe todo el proceso de modelado de clases de objetos en el modelo de objetos, y que además incluye el soporte de las relaciones dinámicas y funcionales entre las clases a través de los modelos dinámico y funcional.

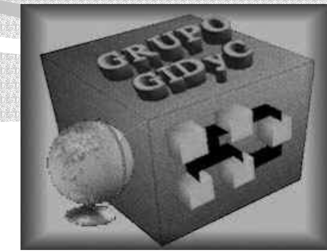
✓ La Técnica de Desarrollo de Sistemas de Objetos (TDSO) está basada en MEDEE y en OMT, en su versión de 1998. Extensión con UML, 2001.

✓ Del primero contiene todas las fases, incluyendo además las de especificación formal [Guttag, 1977 y 1978; Hoare, 1969 y 1987; Crai, 1986].

✓ De la segunda se toman algunos de los diagramas transformados y adaptados para TDSO.

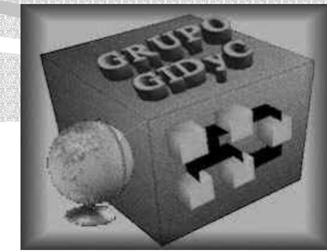
✓ La extensión de tales métodos se hace con la inclusión de una guía para el desarrollo de las pruebas basada en [Jorgensen, 1994; McGregor, 1994; Poston, 1994; en ACM Comm. De 1994].

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA



Orientación por objetos

- Antecedentes: Simula’67
- **Objeto** es la representación de algo que se describe mediante una *estructura* y un *comportamiento*.
- **Estructura:** describe aquellas características de interés presentes en el objeto y que sirven para plasmar el *estado de ese objeto*.
- **Estado:** es el conjunto de valores actuales de sus propiedades.
- **Comportamiento:** está representado por una serie de operaciones, funciones o métodos que modifican o no el estado del objeto, haciendo que ocurra un cambio de estado en el mismo, el cual representa el comportamiento del objeto en la realidad. Así, el comportamiento del objeto está dado por sus cambios de estado.
- Un objeto puede ser conocido y descrito por medio de sus **propiedades** (atributos y relaciones) que son *ilimitados*.
- Un objeto puede componerse de dos o más objetos, conformando así un **objeto compuesto**.
- Cada objeto tiene un único **identificador** que es asignado por el sistema, para aquellos sistemas que soportan la identidad del objeto.



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Orientación por objetos

✓ **Atributos** son las propiedades relevantes de un objeto que representa su estructura. **Simples** o **monovaluados**, en el caso que ellos contengan un único valor a la vez. **Multivaluados** cuando pueden contener varios valores a la vez.

✓ La existencia de un objeto es independiente de los valores de sus atributos.

✓ Dos objetos son **idénticos** si ellos son **el mismo objeto** (sus identificadores son iguales).

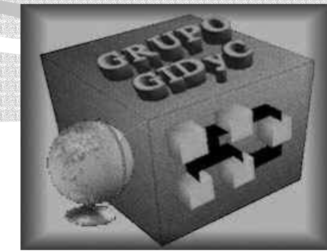
✓ Dos objetos son **iguales** si tienen los mismos valores en sus atributos.

✓ En un sistema o lenguaje que no soporta identidad del objeto la representación gráfica de los objetos es un *árbol*.

✓ En los sistemas que si la soportan es un *grafo*, ya que los objetos compuestos pueden compartir componentes.

✓ Una **operación** es una función asociada al objeto que efectúa una acción atómica sobre el mismo. Tal acción puede o no modificar el estado del objeto.

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA



Tipos abstractos de datos

- TAD se basa en la separación clara entre la implantación y el uso del TAD a través de su interfaz.
- Un TAD tiene una interfaz y puede tener varias implantaciones.
- Definición en dos partes: la especificación y la implementación.
- Una especificación formal es la acción de determinar en términos matemáticos la cosa deseada.
- Un tipo de dato **T** se define "como una clase de valores y una colección de operaciones sobre esos valores. Si las propiedades de esas operaciones son especificadas solamente con axiomas, entonces **T** es un tipo abstracto de dato o una abstracción de dato" [Guttag, 1978].
- Una implantación **correcta** del TAD cumple con todos los axiomas especificados para él.
- Especificación por axiomas algebraicos para el tipo **T** se compone de:
  - **Sintáctica** donde se definen los nombres, dominios y rangos de las operaciones sobre **T**;
  - **Semántica** compuesta del conjunto de axiomas o ecuaciones, que dicen como opera cada una de las operaciones especificadas sobre las otras.
- Implementación se compone de:
  - **Representación**, especifica cómo los valores del TAD serán almacenados en la memoria
  - **Algoritmos,** especifican cómo será usada y manipulada la estructura de datos.
- El acceso al TAD es hecho a través de su interfaz que es visible para los usuarios de ella.
- La implementación del TAD es invisible para el usuario y es visible para el que desarrolla el TAD

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Tipos abstractos de datos

Tipo de dato: Pila[elemento:tipoEle]

Especificación sintáctica:

creaPila() -> Pila,	Crea la pila
meterElePila(Pila,tipoEle) -> Pila,	Inserta un nuevo elemento en el tope
sacarElePila(Pila) -> Pila,	Elimina el elemento que está en el tope
conTopePila(Pila) -> tipoEle \/ {TipoEleNoDef},	Consulta el elemento que está en el tope
vacíaPila(Pila) -> Lógico,	Verifica si la pila está vacía
destruyePila(Pila) -> .	Destruye la pila

Especificación semántica:

Declaración: P: Pila, el: tipoEle;
sacarElePila(creaPila()) = creaPila()
conTopePila(creaPila()) = {TipoEleNoDef}
conTopePila(meterElePila(P,el)) = el
vacíaPila(creaPila()) = Verdadero
vacíaPila(meterElePila(P,el)) = Falso

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Mecanismos de la abstracción de datos

- La abstracción consiste en enfocar los aspectos esenciales inherentes a una entidad e ignorar sus propiedades accidentales.
- El uso de la abstracción preserva la libertad de movimiento para tomar decisiones, ya que evita la acometida prematura de los detalles propios de las implementaciones en los lenguajes de programación.
- **Clasificación:** agrupación de objetos con propiedades y comportamiento similares.
- Una **clase** es un objeto que define la estructura y el comportamiento de un conjunto de objetos que tienen el mismo patrón estructural y de comportamiento.
- Cada objeto que pertenece a una clase es una **instancia** de ella.
- El conjunto de todas las instancias de una clase es la **extensión** de la clase.
- La **instanciación** es el proceso de generación o creación de las instancias de una clase. La definición de una clase normalmente contiene: su nombre, el nombre de sus superclases -clases de las que se hereda-, su interfaz, su estructura y su implementación.
- Las clases formarán una jerarquía denominada **jerarquía de clases**, donde las clases superiores a una clase particular se denominan **superclases** y a la clase particular se le llama **subclase**. La relación entre una superclase y una subclase se denomina ES-UN(A).
- La **metaclase** es la clase que define todas las clases del sistema.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Mecanismos de la abstracción de datos

Las ventajas de la programación orientada por los objetos son las siguientes:

• La abstracción de datos y el ocultamiento de la información aumentan la confiabilidad y ayudan a separar la especificación de la implantación.
• El encadenamiento dinámico incrementa la flexibilidad.
• La herencia junto con el encadenamiento tardío permite la reusabilidad aumentando así la productividad.

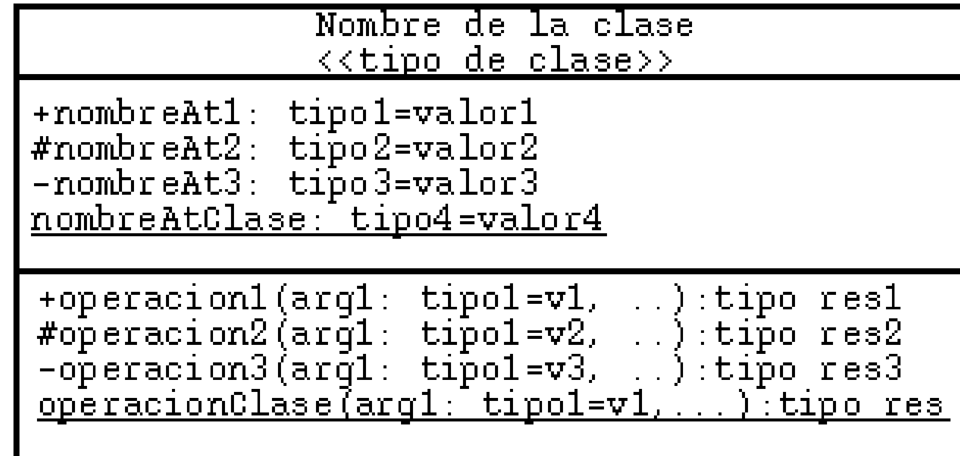
Entre sus desventajas se encuentran:

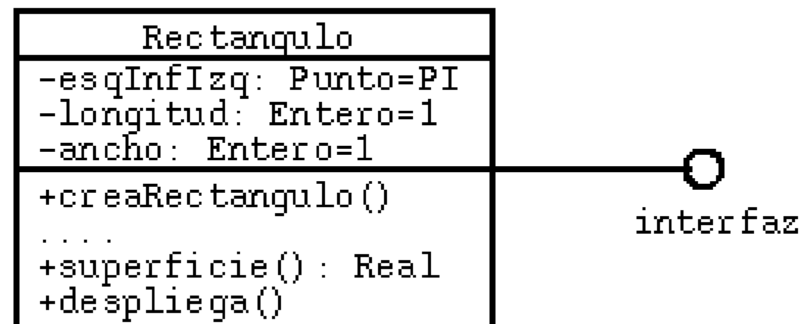
•El costo de tiempo de ejecución del encadenamiento tardío puede llegar a ser importante dependiendo de la aplicación.
• La implantación con lenguajes orientados por objetos es más compleja que con los lenguajes convencionales.
• El programador debe leer con frecuencia extensas librerías de clases.

Notación gráfica de TDSO
Clases

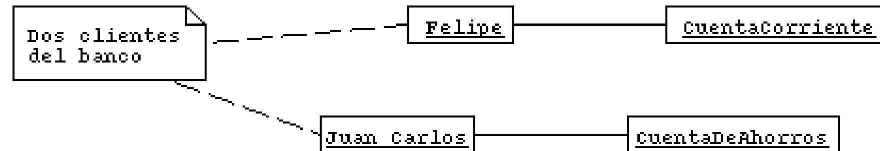
nombreDeLaClase
nombreDelAtributoDeLaInstancia[: tipo [= valorPorOmisión]] $nombreDelAtributoDeLaClase[: tipo [= valorPorOmisión]]
nombreDeLaFunciónDeLaInstancia([listaDeArgumentos]) [: tipoDeRegreso] $nombreDeLaFunciónDeLaClase([listaDeArgumentos]) [: tipoDeRegreso]
nombreDelEventoDeLaInstancia([listaDeFunciones]) $nombreDelEventoDeLaClase([listaDeFunciones])

Notación gráfica de TDSO Clases en UML

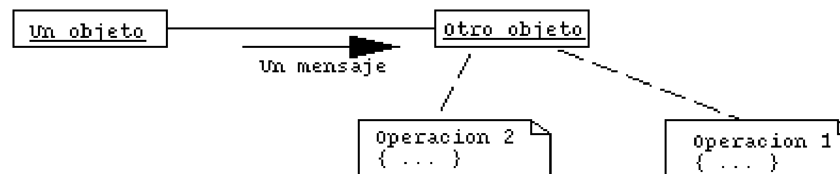




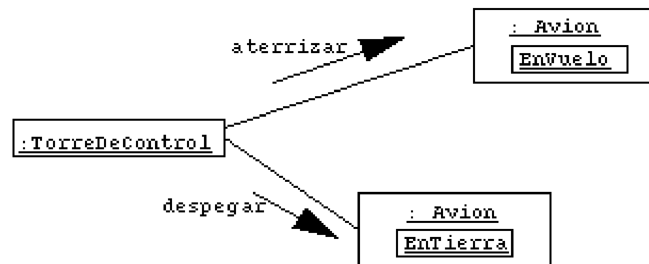
Notación gráfica de TDSO
Objetos y mensajes en UML



Objetos con enlaces



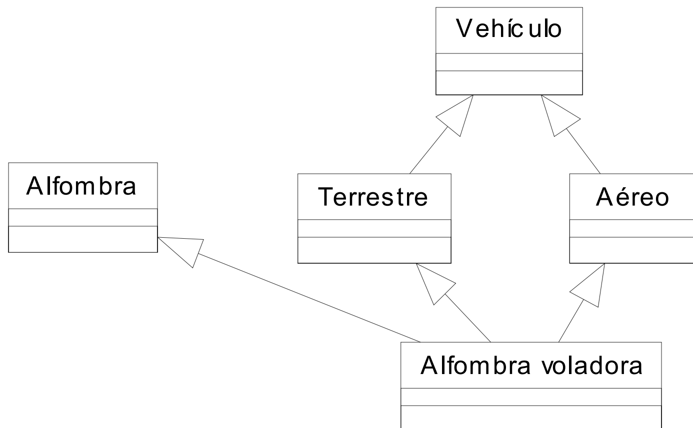
Para responder a un mensaje, el objeto destinatario dispara las operaciones adecuadas que forman parte de su comportamiento.



Enlace entre comportamiento y atributos

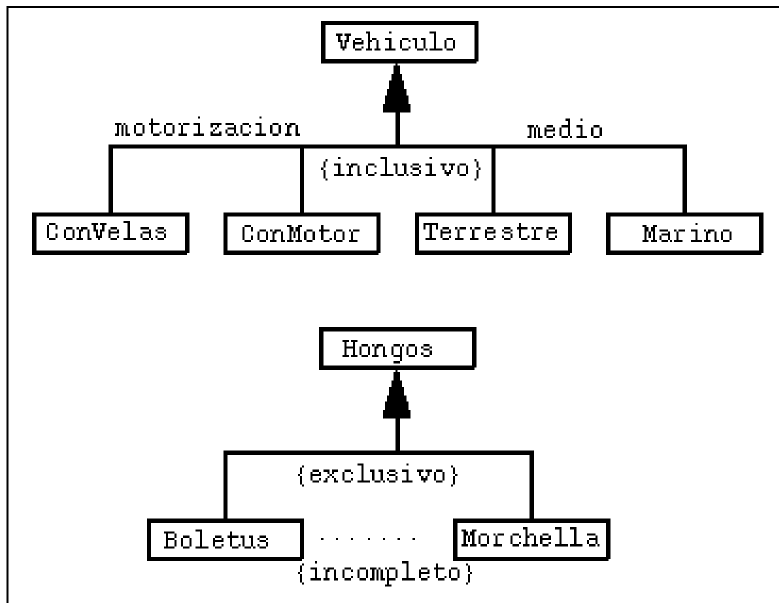
UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Notación gráfica de TDSO
Jerarquía de herencia

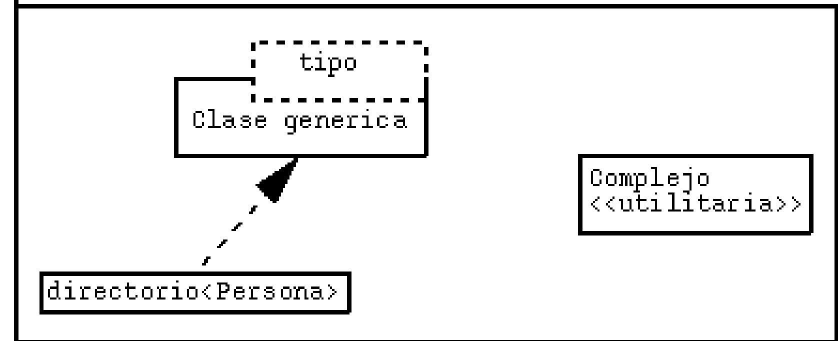



```
Persona
-nombre: Cadena=CadNula
-direccion: Cadena=CadNula
-telefono: Cadena=CadNula
+creaPersona()
....
+desplegar()

Empleado
-cargo: Cadena=CadNula
-sueldo: Real=0.0
-fechaIng: Fecha==Hoy
+creaEmpleado()
....
+desplegar()

Estudiante
-carrera: Carrera=CarNoEsp
-fechaDeInsc: Fecha=Hoy
+creaEstudiante()
....
+desplegar()

Preparador
+creaPreparador()
....
desplegar()
```

Notación gráfica de TDSO
Restricciones de herencia

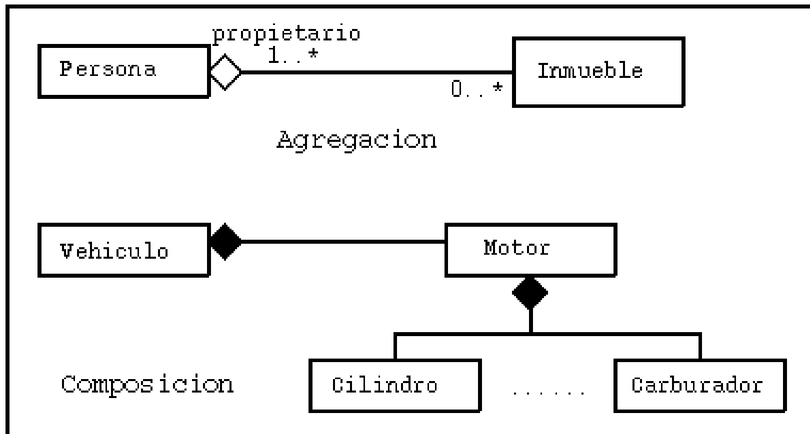


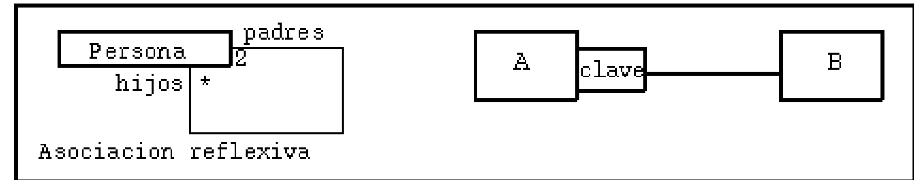
Clases abstractas y utilitarias

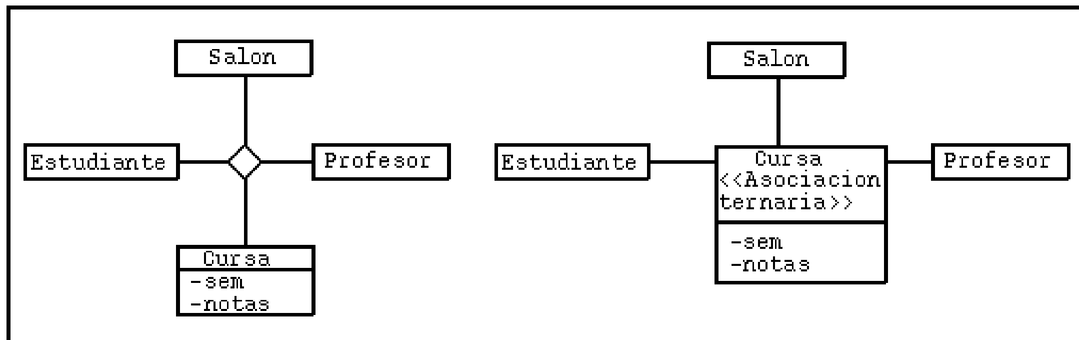


Clases parametrizables y parametrizadas

Notación gráfica de TDSO
Jerarquía de relación



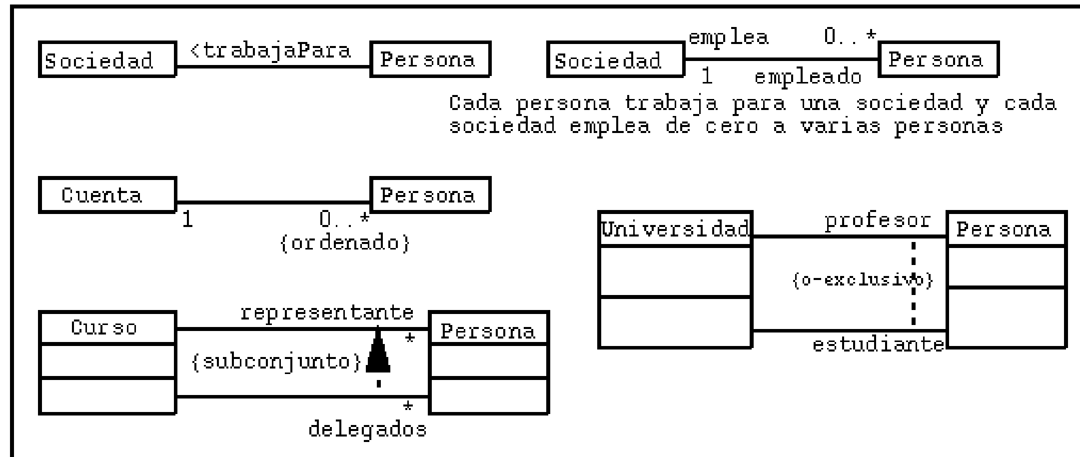




-------------------->

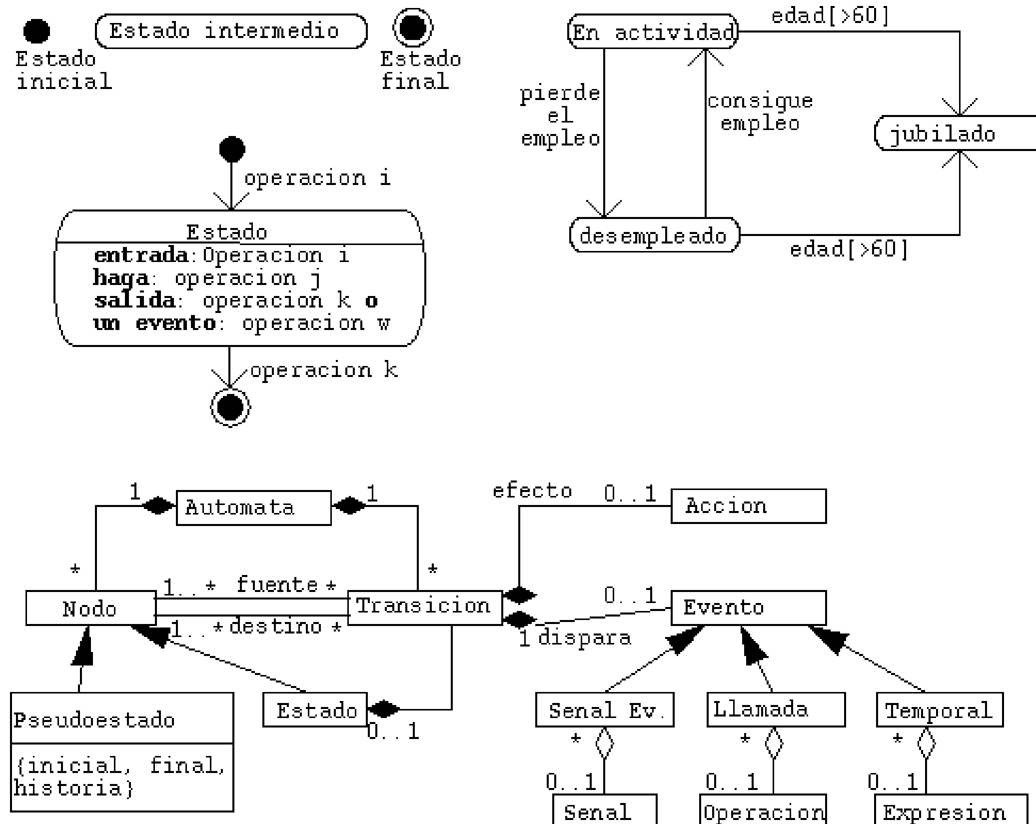
Relación de dependencia

Notación gráfica de TDSO
Jerarquía de relación

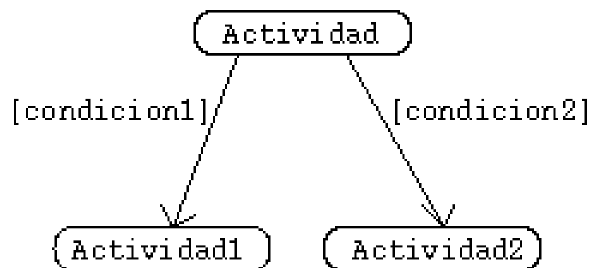


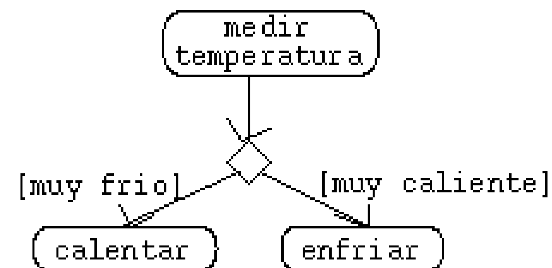
Notación gráfica de TDSO
Dinámica de las clases con transición de estados

Los **eventos** tienen sus **condiciones** de disparo y las **operaciones** que invocan, que producen los **cambios de estado** en instancias de una o más clases. Dos eventos pueden relacionarse o no, en el caso que uno sea continuación de otro o no. Dos eventos no relacionados son **concurrentes**. El tiempo en que ocurre un evento es un atributo implícito, por lo tanto la estructura de la clase lo debe soportar.

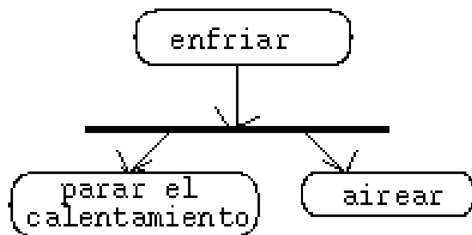


Notación gráfica de TDSO
Dinámica de las clases con actividades

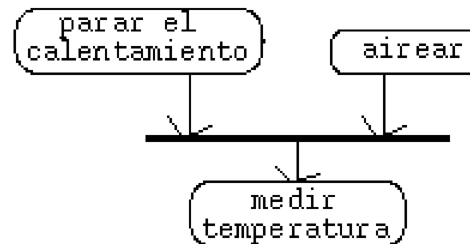




Para representar condiciones y decisiones



Sincronizacion de flujos paralelos

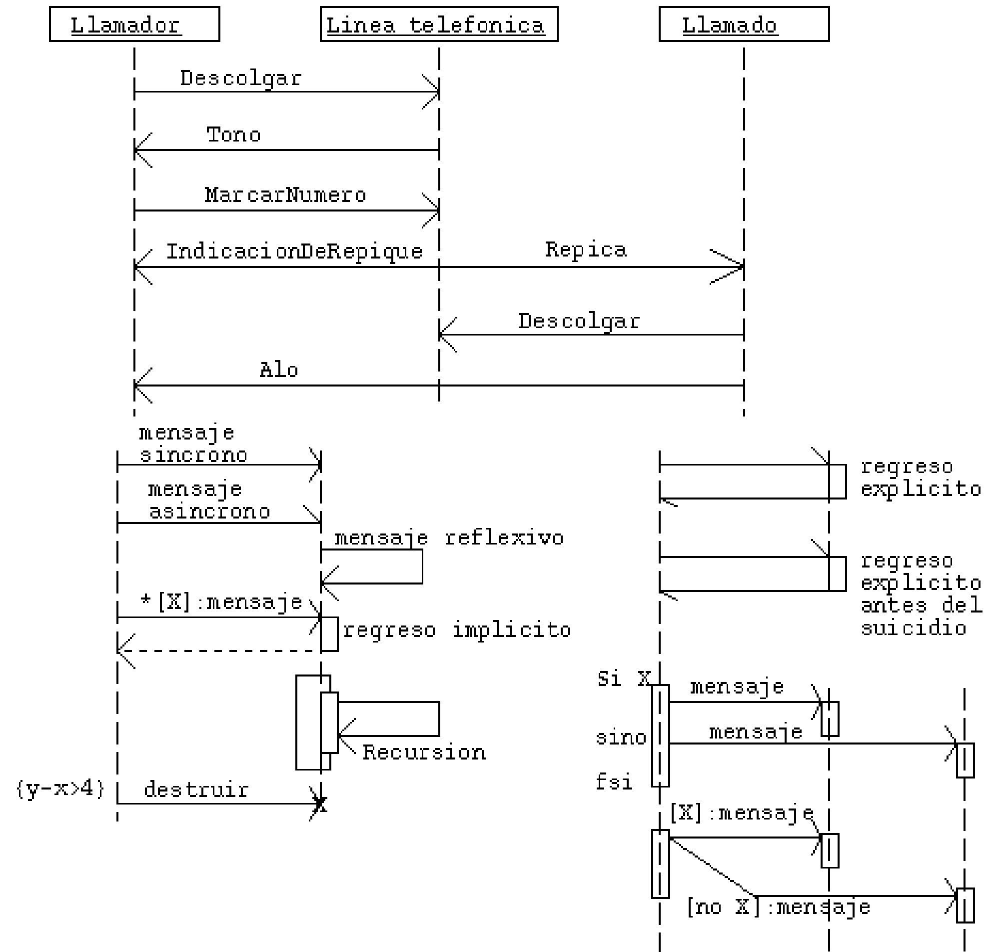


Fusion de flujos paralelos

Notación gráfica de TDSO

Diagramas de secuencia UML

Cuando una función esté compuesta por otras funciones entonces su nombre cambia por el **proceso**, así un proceso es una función de alto nivel que está compuesta por subprocesos u operaciones.

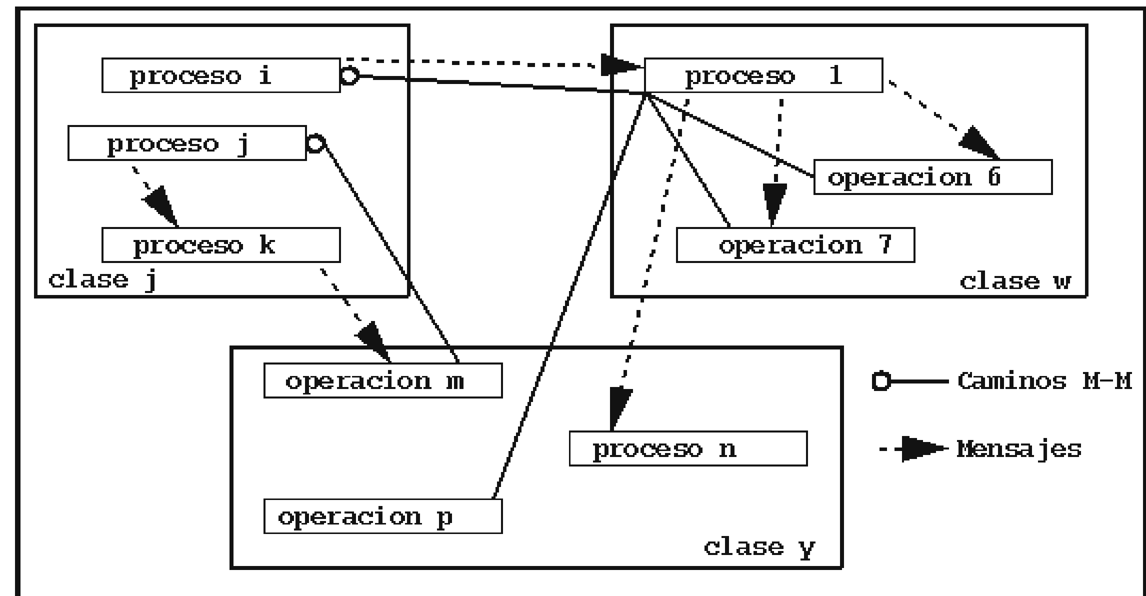


UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Notación gráfica de TDSO
Dinámica con los caminos M-M

La descripción de los procesos del sistema programado se expresa con el diagrama que muestra como un proceso invoca otros procesos u operaciones indicando los caminos método- mensaje (M-M) o caminos de invocación de las funciones.

Un proceso será activado por medio de una invocación al mismo o por medio de uno o más eventos que realicen la invocación ya mencionada.



TDSO

Tomando del método deductivo sus principales conceptos, se tiene que las etapas de la resolución de problemas se expresan en dos partes:

un **universo** de objetos dentro del cual se evoluciona y
un **enunciado** del problema a resolver en ese universo.

Se comienza con un enunciado **Eo** en lenguaje natural, con el que se describe el problema en su primera impresión. A partir de él se profundiza en la comprensión del mismo buscando alternativas de solución, es decir, se sigue un proceso de refinamiento paso a paso expresado por los diferentes enunciados del problema, hasta llegar a una colección de ellos, con la que se completa el enunciado del mismo y así, éste puede ser resuelto.

En este proceso de refinamiento paso a paso, se identifican clases de objetos junto con sus atributos y operaciones, las comunicaciones entre las clases, sus casos de prueba y se aplica la herencia en aquellas clases donde sea conveniente su uso.

TDSO (Eo)

Eo:
Detectar si una cadena de caracteres dada pertenece o no al conjunto de las cadenas de la forma **wCw**$_i$, donde **w** son cadenas formadas en el alfabeto ß={'a','b'} y **w**$_i$ es la cadena inversa.

Se plantea el problema como un sistema de ecuaciones explícitas o definiciones formales, donde **n** de ellas pueden escribirse como: **X1,...,Xn = e1,...,en**, donde el identificador **Xi** está asociado a la expresión **ei**.
Cuando los enunciados **Ei** son muy complejos, ellos deben ser divididos y se obtiene un conjunto de ecuaciones por subproblema denominados **módulos** o rutinas.
Una rutina **f** puede ser una función en el caso que se defina un único valor de **X1,...,Xp**.
El módulo que no sirve para definir otros se llama **principal** o programa.
Cada módulo se escribirá con un encabezado, un léxico o definiciones informales de los **Xi** y un sistema de ecuaciones o definiciones formales.

Especificación TDSO

ENCABEZADO		
#	ECUACIONES (Definiciones formales)	LEXICO (Definiciones informales)

TDSO. Ecuaciones.

ASIGNACIONES	
Variables = Expresiones	X1,...,Xn = E1,...,En
DEFINICIÓN CONDICIONAL	
SIMPLE	X1,...,Xn = **Si** (C) **entonces** E1,...,En **sino** E'1,...,E'n **fsi**
MÚLTIPLE	**En caso que** Xi=E1: X'1,...,X'n=E'1,...,E'n ….. **en otro caso** Ej: Y'1,...,Y'm=F'1,...,F'm

TDSO. Lazos.

LAZOS	
Repita para **j** desde **Vi** hasta **Vf** con incremento **inc**	(X1,...,Xn = E1,...,En) **j =** Vi, Vf, inc
Repita mientras **C** se cumpla	**(C)** [X1,...,Xn = E1,...,En]
Repita hasta que **C** se cumpla	[X1,...,Xn = E1,...,En] **(C)**

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

TDSO. Entradas.

ENTRADAS	
Lectura desde el teclado	X1,...,Xn = valor suministrado
Lectura desde un archivo Aj de acceso secuencial	X1,...,Xn = registro siguiente de Aj
Lectura desde un archivo Aj de acceso directo	X1,...,Xn = registro k de Aj
Lectura desde un archivo Aj de acceso indizado o aleatorio según una **clave**	X1,...,Xn = registro de Aj según **clave**=valor

TDSO. Salidas y rutinas.

SALIDAS	
Escritura en pantalla	Despliegue X1,...,Xn
Escritura en papel	Imprima X1,...,Xn
Escritura en archivo Aj de acceso secuencial	Escriba X1,...,Xn en el registro siguiente de Aj
Escritura en archivo Aj de acceso directo	Escriba X1,...,Xn en el registro k de Aj
Escritura en archivo Aj de acceso indizado o aleatorio según una **clave**	Escriba X1,...,Xn en el registro de Aj según **clave**=valor
RUTINAS	
Funciones	Xj = objeto.operación(parámetros)
Procedimientos	objeto.operación(parámetros)

TDSO. Universo del programa.

<table>
<tr><td colspan="3">fecha
Universo nombreDelPrograma</td></tr>
<tr><td>#</td><td>Definiciones formales de las clases o tipo</td><td>Definiciones informales de las clases o tipo</td></tr>
</table>

- Para la definición del universo del programa se utilizan los tipos abstractos de datos.
- Por ello, se deben definir allí los objetos utilizados en el programa.
- La selección de las clases de objetos es un paso que influye en la estructura y en el desempeño del programa.
- Para iniciar este paso es aconsejable partir de los documentos de requerimientos y el proceso se hace descendentemente o de arriba-hacia-abajo

TDSO. Especificaciones algebraicas.

<table>
<tr><td colspan="3">fecha
Especificación (#) nombreDelTipo o Clase</td></tr>
<tr><td>1
2
3</td><td>Sintáctica:
Declaraciones:
Semántica:</td><td>Definiciones informales de las operaciones de la clase o del tipo.</td></tr>
</table>

Para las clases o tipos generados y usados en el problema se realizan sus respectivos enunciados **Ei**, colocando las funciones y los axiomas y precondiciones, y a ello se le denomina especificaciones algebraicas.
La especificación algebraica debe ser consistente y completa.
Ella es **consistente** si no altera las especificaciones de los tipos que utiliza.
Es suficientemente **completa**, si para todo término sin variables de la forma s(a1,...,ap), donde s es el selector, puede ser reescrito, gracias a los axiomas, en un término que no contiene ninguna de las operaciones del tipo especificado, es decir, hay suficientes axiomas para salir de ese tipo.

TDSO. Operaciones.

- Las operaciones definidas sobre la clase se catalogan como: constructores, observadores, transformadores, convertidores y destructores.
- Un **constructor** es una operación que crea un objeto de ese tipo o clase y eventuelmente, puede iniciarlo con algún valor por omisión, lo cual es altamente recomendable en el momento de la implementación.
- Un **observador** es una operación que no modifica el estado del objeto y eventualmente puede mostrar el contenido de alguno de sus atributos.
- Un **transformador** es aquella operación que modifica el estado del objeto y que eventualmente, puede realizar algún cálculo y regresar algún valor que indique si el cambio se efectuó exitosamente.
- Un **convertidor** es una función que crea un objeto nuevo a partir de otro objeto que pertenece a otra clase diferente de la clase del objeto nuevo.
- Un **destructor** es una operación que destruye el objeto.
- Se especifican las operaciones de la clase como: operación, proceso'P' o evento 'E'

TDSO. Ejemplo.

E1:

Construir un programa que detecte si una cadena de caracteres dada por el usuario, con longitud máxima de 255 caracteres, es una cadena que pertenece a la familia de las cadenas que tienen como forma general: **wCwi**. Donde **w** es una cadena formada con la concatenación de los símbolos del alfabeto ß={'a','b'}, **wi** es la cadena inversa de **w** y **C** es un caracter que no pertenece a ß.

- El tipo de función: 'P' para los procesos, 'E' para los eventos y sin tipo para el resto de las operaciones.
- El tipo de acceso: 'R' para las funciones privadas, 'O' para las protegidas y sin tipo para las públicas

<table>
<tr><td colspan="3">fecha
(#Clase.#Función) (tipoDeFunción.tipoDeAcceso)
nombreDeLaFunción(Tipo: parámetro,...)[:tipoDeResultado]
{ pre: precondiciones } { pos: poscondiciones }</td></tr>
<tr><td>#</td><td>Algoritmo</td><td>Documentación 2</td></tr>
<tr><td>#</td><td>Casos de prueba</td><td>Documentación de pruebas</td></tr>
</table>

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

TDSO. Ejemplo 0.

E2:

10-06-01		
	(0) programa familia **pos: { detecta si la cadena pertenece o no a la familia wCw^i }**	
1 3 2 4	cad=valor suministrado per=Si (Lon(cad) es impar y cad contiene una 'C' y cad < > "") entonces TratarCadena(cad) fsi per=Falso Si (per o cad = "") entonces Despliegue "Cadena válida" sino Despliegue "Cadena inválida" fsi	**-cad:** Cadena: Cadena leida. **-per:** Lógico: Es verdadero si cad pertenece a la familia deseada. **-Lon():** Operación del tipo Cadena. **-TratarCadena():** Función del programa familia que efectua el tratamiento de cad, devolviendo verdadero si cad pertenece a la familia.
1	cad='aaCaa' => Cadena válida	Cadena válida
2	cad='abbCba' => Cadena inválida	Inválida en longitud
3	cad='aaa' => Cadena inválida	Inválida por falta de C
4	cad = ‘’ =>Cadena válida	Válida por ser nula
5	cad = ‘asCaa’ =>Cadena inválida	Inválida por el alfabeto
6	cad = ‘baCba’ =>Cadena inválida	Inválida en secuencia

TDSO. Ejemplo 0.1.

27/3/95

(0.1) (P)TratarCadena(Cadena cad):Lógico

{ pre: cad < > 0 } { pos: Regresa Verdadero si cad pertenece a la familia wCwi }

1 2 4 3	w = Sub(cad, 1, Ind(cad, 'C')) wi = Sub(cad, Ind(cad, 'C') + 1) TratarCadena = Si (Lon(w) = Lon(wi)) entonces Verificar cada caracter de w y wi en ('a' , 'b') Compare wj con wik , si son iguales incremente j y decremente k sino regrese fsi TratarCadena=Falso	**-w:** Cadena: Cadena antes del caracter 'C'. **-wi:** Cadena: Cadena después del caracter 'C'. **-j:** Entero: Posición de un caracter en w. **-k:** Entero: Posición de un caracter en wi. **-Sub(), Ind(), Lon():** Operaciones del tipo Cadena
1	cad='aaCaa' => TratarCadena = Verdadero	Cadena válida
2	cad = 'asCaa' => TratarCadena = Falso	2do. Caracter fuera dell alfabeto
3	cad = 'baCba' => TratarCadena = Falso	Secuencias iguales

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

TDSO. E3 (0).

10-06-97		
	(0) programa familia	
	pos: { detecta si la cadena pertenece o no a la familia wCw^i }	
1 3 2 4	cad=valor suministrado per=Si (Mod(Lon(cad), 2) < > 0 /\ Ind(cad, 'C')< > 0 /\ cad < > "") entonces TratarCadena(cad) fsi per=Falso Si (per \/ cad = "") entonces Despliegue "la cadena pertenece a la familia" sino Despliegue "la cadena no pertenece a la familia" fsi	**-cad:** Cadena: Cadena leida. **-per:** Lógico: Es verdadero si cad pertenece a la familia deseada. **-Lon(), Ind():** Operaciones del tipo Cadena. **-TratarCadena():** Función que efectúa el tratamiento de cad, devolviendo verdadero si cad pertenece a la familia. **-Mod():** Función que regresa el resto de la división.
1	cad='aaCaa' => la cadena pertenece a la familia	Cadena válida
2	cad='abbCba' => la cadena no pertenece a la familia	Inválida en longitud
3	cad='aaa' => la cadena no pertenece a la familia	Inválida por falta de C
4	cad = '' => la cadena pertenece a la familia	Válida por ser nula
5	cad = 'asCaa' => la cadena no pertenece a la familia	Inválida por el alfabeto
6	cad = 'baCba' => la cadena no pertenece a la familia	Inválida en secuencia

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

TDSO. E3 (0.1).

27/3/95

(0.1) (P)TratarCadena(Cadena cad):Lógico

{ pre: cad < > /\ }   { pos: Regresa Verdadero si cad pertenece a la familia wCwi }

4	(Sub(cad, j, 1) < > 'C' /\ Sub(cad, j, 1) ∈ {'a', 'b' })
	    [p.meter(Sub(cad, j, 1))
	     j = j + 1]
		-car: Cadena: Cadena que está actualmente en tratamiento.

3	j = 1	**-p:** Pila: Almacena los caracteres antes de la 'C'.
5	(Sub(cad, j, 1) = car /\ j <= Lon(cad) /\ ¬p.vacia( )) [j = j + 1 car = p.tope( ) p.sacar( )]	**-j:** Entero: Posición de un caracter en cad. **-meter( ), vacia( ), tope( ), sacar( ):** Operaciones del tipo Pila.
2	TratarCadena=Falso	**-Sub(), Lon():** Operaciones del tipo Cadena
1	Pila p	
6	TratarCadena = Si (j > Lon(cad) /\ p.vacia( )) entonces Verdadero fsi	
1	Cad = 'aaCaa' => TratarCadena = Verdadero	Cadena válida
2	cad = 'asCaa' => TratarCadena = Falso	2do. Caracter fuera dell alfabeto
3	cad = 'baCba' => TratarCadena = Falso	Secuencias iguales

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

TDSO. Implementación de una clase.

<table>
<tr><td colspan="3">27/3/97
Implementación TAD(#) (tipoDeClase)Clase:nombreDeLaClase
Clases: listaDeClasesYaDefinidas</td></tr>
<tr><td>1
2

3</td><td>Superclases:
Estructura:
Atributos de las instancias:
Atributos de la clase:
Funciones:
sobre las instancias:
sobre la clase:</td><td>Descripción de cada superclase, atributo y función, catalogadas en constructores, convertidores, observadores, transformadores,
eventos, procesos y destructor.
(Los tipos de los atributos y las superclases deben ser clases ya definidas e incluidas en la lista)</td></tr>
</table>

TDSO. Universo familia.

<table>
<tr><td colspan="3">27-03-95</td></tr>
<tr><td colspan="2" align="right">Universo</td><td>familia</td></tr>
<tr><td>1

2
3
4
5
6</td><td>NodoPila: Estructura
el: Caracter
sig: ApNodoPila
fin
ApNodoPila: Apuntador de NodoPila
Cardinal: Entero+
Cadena
Entero
Pila[el: tipoEle]</td><td>-Cardinal: tipo: Subconjunto de los números enteros positivos, incluyendo el cero. (tipo básico)
-Pila[el: tipoEle]: tipo: Pila
-Cadena: tipo: Cadena de caracteres. (tipo básico)
-Entero: Tipo básico.
-NodoPila: tipo: Tipo compuesto soportado por la mayoría de los lenguajes. (tipo básico)
-ApNodoPila: tipo: Dirección de un dato tipo NodoPila</td></tr>
</table>

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

TDSO. Especificación de Pila.

27/3/97 **Especificación (4) Pila[el: tipoEle]**		
1	**Sintáctica**: creaPila()-> Pila, meterElePila(Pila,tipoEle) -> Pila, sacarElePila(Pila) -> Pila, conTopePila(Pila) -> tipoEle, vacíaPila(Pila) -> Lógico, destruyePila(Pila) -> .	**-creaPila():** Crea una pila vacía. **-meterElePila():** Ingresa un nuevo elemento a la pila por el tope de la misma. **-sacarElePila():** Elimina el elemento que está actualmente en el tope de la pila. Si la pila está vacía no hace ninguna eliminación. **-vacíaPila():** Regresa Verdadero si la pila está vacía. **-destruyePila():** Destruye la pila. **-conTopePila():** Devuelve el elemento que se encuentra actualmente en el tope de la pila. Si la pila está vacía devuelve un valor especial que lo indica.
2	**Declaraciones**: p: Pila e: tipoEle	
3	**Semántica**: vacíaPila(creaPila())=Verdadero vacíaPila(meterElePila(creaPila(),e))=Falso conTopePila(creaPila())={TipoEleNoDef} sacarElePila(creaPila())=creaPila()	

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

TDSO. Implementación de Pila.

<table>
<tr><td colspan="3">27/3/97
Implementación TAD(4) Clase: Pila
Clases: Caracter, Lógico, NodoPila</td></tr>
<tr><td>1
2

3</td><td>Superclases: Ninguna
Estructura:
tope: ApNodoPila
Operaciones:
creaPila()
meterElePila(e: Caracter)
sacarElePila()
conTopePila():Carácter
vacíaPila():Lógico
destruyePila()</td><td>-tope: Apuntador al nodo que está en el tope de la pila.
-creaPila(): Constructor e iniciador de la pila.
-meterElePila(): Transformador que inserta un nuevo elemento en la pila.
-sacarElePila(): Transformador que elimina el elemento en el tope de la pila, si él existe.
-conTopePila(): Observador que regresa una copia del elemento en el tope de la pila.
-vacíaPila(): Observador que regresa Verdadero si la pila está vacía y Falso en caso contrario.
-destruyePila(): Destructor.</td></tr>
</table>

TDSO. Implementación de Pila en C++.

<table>
<tr><td colspan="3">27/3/97
Implementación TAD(4) Clase: Pila
Clases: char, int, NodoPila</td></tr>
<tr><td>1
2

3</td><td>Superclases: Ninguna
Estructura:
*tope: NodoPila
Operaciones:
Pila( )
void meter(char e)
void sacar( )
char consultar( )
int vacia( )
~Pila( )</td><td>-tope: Apuntador al nodo que está en el tope de la pila.
-Pila(): Constructor e iniciador de la pila.
-meter(): Transformador que inserta un nuevo elemento en la pila.
-sacar(): Transformador que elimina el elemento en el tope de la pila, si él existe.
-consultar(): Observador que regresa una copia del elemento en el tope de la pila.
-vacia(): Observador que regresa Verdadero si la pila está vacía y Falso en caso contrario.
- ~Pila(): Destructor.</td></tr>
</table>

TDSO. Pila.

27/3/97 **(4.1) creaPila( )** **{ pos: Crea la pila vacía. tope = Nulo }**		
1	tope = Nulo	**-tope:** Definido en TratarCadena

27/3/97 **(4.1) Pila( )** **{ pos: Crea la pila vacía. tope = Nulo }**		
1	tope = 0;	**-tope:** Definido en TratarCadena

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

TDSO. meterElePila( ).

<table>
<tr><td colspan="3">27/3/97
(4.2) meterElePila(e: Caracter)
{ pre: e < > ‘255’ } { pos: P’ = P + e }</td></tr>
<tr><td>1</td><td>Si (e < > ‘255’) entonces
p = nuevo NodoPila
p.el = e
p.sig = tope
tope = p
sino
despliegue ‘Elemento inválido’
fsi</td><td>-tope. Definido en Pila.
-p: ApNodoPila. Variable auxiliar para contener la información del nodo nuevo.
-el, sig. Definidos en NodoPila.
-e: Caracter. Nuevo elemento que se inserta en la pila, debe ser diferente del caracter No Definido ‘255’.</td></tr>
<tr><td>1</td><td>e = ‘255’ => Elemento inválido</td><td>El elemento no cumple la precondición</td></tr>
<tr><td>2</td><td>e = ‘k’ => P = P + ‘k’</td><td>El nuevo elemento es el tope de la pila</td></tr>
</table>

TDSO. meter( ).

27/3/97

(4.2) meter(char e)

{ pre: e < > ‘255’ } **{ pos: P’ = P + e }**

1	if (e != ‘255’)
	{ p = new NodoPila;
	p->el = e;
	p->sig = tope;
	tope = p;
	}
	else
	cout << ”Elemento inválido”;
		-tope. Definido en Pila.
		-p: ApNodoPila. Variable auxiliar para contener la información del nodo nuevo.
		-el, sig. Definidos en NodoPila.
		-e: Caracter. Nuevo elemento que se inserta en la pila, debe ser diferente del caracter No Definido ‘255’.

1	e = ‘255’ => Elemento inválido	El elemento no cumple la precondición
2	e = ‘k’ => P = P + ‘k’	El nuevo elemento es el tope de la pila

TDSO. sacarElePila( ).

27/3/97

(4.3) sacarElePila( )

{ pre: < > vaciaPila( ) } **{ pos: P' = P - e }**

1	Si (tope = Nulo) entonces
	  despliegue 'Pila vacía'
	sino
	  r = tope
	  tope = tope.sig
	  regrese r
	fsi
		-tope. Definido en Pila.
		-r: ApNodoPila. Variable auxiliar para guardar la dirección del nodo tope.
		-sig. Definidos en NodoPila.

1	tope = Nulo => Pila vacía	No cumple la precondición
2	tope < > Nulo => P = P - tope.el	La pila disminuyó en un elemento

TDSO. sacar( ).

<table>
<tr><td colspan="3">27/3/97
(4.3) sacar( )
{ pre: < > vaciaPila( ) } { pos: P’ = P - e }</td></tr>
<tr><td>1</td><td>if (tope != 0)
cout << “Pila vacía”;
else
{ r = tope;
tope = tope->sig;
delete r;
};</td><td>-tope. Definido en Pila.
-r: ApNodoPila. Variable auxiliar para guardar la dirección del nodo tope.
-sig. Definidos en NodoPila.</td></tr>
<tr><td>1</td><td>tope = Nulo => Pila vacía</td><td>No cumple la precondición</td></tr>
<tr><td>2</td><td>tope < > Nulo => P = P - tope.el</td><td>La pila disminuyó en un elemento</td></tr>
</table>

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

TDSO. conTopePila( ).

27/3/97 **(4.4) conTopePila( ): Caracter** **{ pos: P’ = P }**		
1	si (tope = Nulo) entonces regrese ‘255’ sino regrese tope.el fsi	**-tope.** Definido en Pila. **-el.** Definidos en NodoPila. ‘255’ es el valor indefinido del tipo Caracter.
1	tope = Nulo => ‘255’	Regresa el elemento indefinido
2	tope < > Nulo => tope.el	Regresa el elemento que está en el tope

TDSO. consultar( ).

<table>
<tr><td colspan="3">27/3/97
(4.4) consultar( ): char
{ pos: P’ = P }</td></tr>
<tr><td>1</td><td>if (tope == 0)
return ‘255’;
else
return (tope->el);</td><td>-tope. Definido en Pila.
-el. Definidos en NodoPila. ‘255’ es el valor indefinido del tipo Caracter.</td></tr>
<tr><td>1</td><td>tope = Nulo => ‘255’</td><td>Regresa el elemento indefinido</td></tr>
<tr><td>2</td><td>tope < > Nulo => tope.el</td><td>Regresa el elemento que está en el tope</td></tr>
</table>

TDSO. vaciaPila( ).

27/3/97 **(4.5) vaciaPila( ): Lógico** **{ pos: P' = P }**		
1	regrese (tope = Nulo)	**-tope.** Definido en Pila.
1	tope = Nulo => Verdadero	Regresa verdadero pues la pila está vacía
2	tope < > Nulo => Falso	Regresa falso, pues la pila no está vacía

27/3/97 **(4.5) vacia( ): int** **{ pos: P' = P }**		
1	return (tope = = 0);	**-tope.** Definido en Pila.
1	tope = Nulo => Verdadero	Regresa verdadero pues la pila está vacía
2	tope < > Nulo => Falso	Regresa falso, pues la pila no está vacía

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

TDSO. destruyePila( ).

27/3/97 **(4.6) destruyePila( )** **{ pos: Deja la pila vacía }**		
1	(tope < > Nulo) [sacarElePila( )]	**-tope, sacarElePila( ).** Definido en Pila.

27/3/97 **(4.6) ~Pila( )** **{ pos: Deja la pila vacía }**		
1	while(tope != 0) sacar( );	**-tope, sacar( ).** Definido en Pila.
2	return;	

A la implementación presentada deberían hacérsele ciertos cambios para que dicha implementación pueda ser usada con cualquier tipo de elemento, es decir una pila genérica, ¿Qué cambios puede usted sugerir?,¿Puede usted probarlos?.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Pruebas orientadas por objetos

1. **Pruebas de cada operación:** especificadas en G. J. Myers [Myer-83], pero cuyos casos de prueba, específicos para la operación que se está elaborando, serán colocados en la misma forma de la definición de la operación, especificando en el centro las descripciones formales de cada caso de prueba, bajo la forma: entradas(X1=c1,...,Xn=cn) => salida(Xi=ci,...,Xp=cp) en el lado izquierdo la enumeración deseada para la realización de tales pruebas y en el lado derecho la descripción informal de cada caso de prueba.
2. **Pruebas del estado final de cada mensaje:** para lo cual se debe utiliza el gráfico de los procesos, donde se muestran los caminos M-M, que son una secuencia de ejecuciones de operaciones encadenadas por los mensajes. Un camino método-mensaje se inicia en un proceso y termina en una operación, denominándose a este punto, el estado final del mensaje. Este tipo de prueba se especifica en la misma forma.
3. **Pruebas del funciones atómicas del sistema o programa:** Una función atómica del sistema se inicia en un puerto de entrada seguido por un conjunto de caminos método-mensaje y terminando en un puerto de salida de la función mencionada. En nuestro caso son las pruebas de los procesos mencionadas anteriormente, incluyendo además los eventos.

UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Pruebas orientadas por objetos con herencia de clases

- Cuando el programa involucra varias clases o tipos y existe herencia entre algunas de ellas, según J. McGregor y T. Korson en [McKo-94], se debe tener en cuenta adicionalmente que:

  - las precondiciones de cada método de una subclase no deben ser más fuertes que las de sus superclases.

  - las poscondiciones de cada método de una subclase no deben ser más débiles que las de sus superclases.

  - la invariante de una subclase es un superconjunto de las invariantes de sus superclases.

Gráfico de los procesos del programa

TratarCadena
familia
vaciaPila
destruyePila
conTopePila
sacarElePila
meterElePila
creaPila
Pila
Mod
Entero
Sub
Lon
Ind
Cadena
Caminos M-M
Mensajes

Conclusiones

- Permite modelar las propiedades estáticas y dinámicas de los objetos involucrados en el sistema.

- Se sigue manteniendo la descomposición modular-jerárquica de los problemas y el refinamiento paso a paso.

- Se construye, a medida que se analiza el problema, la documentación y las pruebas del mismo, en forma clara, modular y jerárquica.

- Se incluyen los casos de prueba, no presentes en el MEDEE original.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Conclusiones (1)

- La especificación se hace con miras a la utilización de herramientas de programación orientadas por objetos, aunque no imposibilita el uso de las que no lo sean.

- La secuencia de análisis, especificación, diseño, documentación y pruebas en varias etapas hace que dichos procesos se sustenten en bases más seguras, donde sea poco factible el olvido de algún detalle importante.

- Permite abordar soluciones tanto iterativas como recursivas.

- Permite cierta libertad a la hora de incluir detalles olvidados en alguna función, así como la libertad de comenzar el desarrollo de los módulos por aquel aspecto que primero viene a la mente, para una vez finalizado, hacer la secuenciación de las ecuaciones o estructuras ya definidas