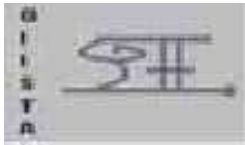


Lógica y programación orientada a objetos: Un enfoque basado en problemas





Grupo de Investigación
en Ingeniería de Software
del Tecnológico de Antioquia

Proyecto Sismoo:
Sistema para el modelamiento por objetos

Investigadores principales:
RICARDO DE JESÚS BOTERO TABARES
CARLOS ARTURO CASTRO CASTRO
GABRIEL ENRIQUE TABORDA BLANDÓN

Coinvestigadores:
JUAN DAVID MAYA MONTOYA
MIGUEL ÁNGEL VALENCIA LÓPEZ

Ilustrador:
JAIME ROLDÁN ARIAS

ISBN: 978-958-8628-00-4

Contenido

PRESENTACIÓN.....1

Capítulo 1. Una didáctica para el aprendizaje de la programación

1.1. ALGUNAS CONSIDERACIONES PEDAGÓGICAS PARA EL APRENDIZAJE DE LA PROGRAMACIÓN.....11

1.2. FASES PARA LA SOLUCIÓN DE UN PROBLEMA14

1.3. TIPO DE PROBLEMAS A TRATAR21

1.4. REFERENCIAS24

Capítulo 2. Fundamentos de programación orientada a objetos

2.1. EL MUNDO DE LOS OBJETOS, EL MUNDO DE LAS COSAS.....31

 LOS CONCEPTOS DE OBJETO Y CLASE32

 PROBLEMA 1: HOLA MUNDO.....37

 CREACIÓN DE OBJETOS Y PASO DE MENSAJES42

 PROBLEMA 2: OBJETOS TIPO CÍRCULO.....43

2.2. TIPOS DE DATOS PRIMITIVOS Y VARIABLES47

2.3. OPERADORES Y EXPRESIONES51

2.4. CLASES DE USO COMÚN: UN MICRO MUNDO PARA EFECTOS DE REUTILIZACIÓN58

TIPOS DE DATOS ESTÁNDAR COMO OBJETOS58

LA CLASE ENTERO

LA CLASE REAL.....

LA CLASE CHARACTER

LA CLASE LOGICO

PAQUETES DE USO COMÚN.....65

2.5. CONFIGURACIÓN DE LAS CLASES DE USO COMÚN.....67

LA CLASE OBJETO.....67

LA CLASE TIPODE DATO.....68

LA CLASE FLUJO69

LA CLASE CADENA.....70

LA CLASE MAT71

2.6. PROBLEMAS RESUELTOS CON LAS CLASES DE USO COMÚN73

PROBLEMA 3: CLASES DE USO COMÚN75

PROBLEMA 4: CONVERSIÓN DE UNA CONSTANTE A DIFERENTES BASES NUMÉRICAS79

2.7. EJERCICIOS PROPUESTOS.....82

2.8. REFERENCIAS84

Capítulo 3. Clases: tipos de datos abstractos

3.1. ESTRUCTURA GENERAL DE UNA CLASE.....91

3.2. MÉTODOS94

DEFINICIÓN DE UN MÉTODO95

INVOCACIÓN DE UN MÉTODO.....97

MÉTODOS ESTÁTICOS98

PROBLEMA 5: OPERACIONES CON UN NÚMERO ENTERO.....99

PASO DE PARÁMETROS POR VALOR VS. PASO DE PARÁMETROS POR REFERENCIA.....105

3.3. SOBRECARGA DE MÉTODOS.....106

PROBLEMA 6: EMPLEADO CON MAYOR SALARIO.....107

PROBLEMA 7: INFORME SOBRECARGADO.....112

3.4.	ESTRUCTURAS DE CONTROL.....	117
	LA ESTRUCTURA SECUENCIA	118
	LA ESTRUCTURA SELECCIÓN	118
	PROBLEMA 8: PRODUCTO MÁS CARO	121
	PROBLEMA 9: ESTADÍSTICAS POR PROCEDENCIA.....	127
	LA ESTRUCTURA ITERACIÓN	133
	PROBLEMA 10: PRIMEROS CIENTO NATURALES.....	139
3.5.	BANDERA O INTERRUPTOR.....	142
	PROBLEMA 11: SUCESIÓN NUMÉRICA	142
3.6.	MÉTODOS RECURSIVOS.....	145
	PROBLEMA 12: FACTORIAL DE UN NÚMERO	146
	PROBLEMA 13: CÁLCULO DE UN TÉRMINO DE FIBONACCI	149
3.7.	EJERCICIOS PROPUESTOS.....	153
3.8.	REFERENCIAS	156

Capítulo 4. Arreglos

4.1.	OPERACIONES CON ARREGLOS	161
	DECLARACIÓN DE UN ARREGLO.....	162
	ASIGNACIÓN DE DATOS A UN ARREGLO	164
	ACCESO A LOS ELEMENTOS DE UN ARREGLO	165
	PROBLEMA 14: SUCESIÓN NUMÉRICA ALMACENADA EN UN VECTOR	165
4.2.	LA CLASE VECTOR	168
	PROBLEMA 15: UNIÓN DE DOS VECTORES.....	176
	PROBLEMA 16: BÚSQUEDA BINARIA RECURSIVA	179
4.3.	LA CLASE MATRIZ	182
	PROBLEMA 17: PROCESO ELECTORAL	189
4.4.	EJERCICIOS PROPUESTOS.....	196
4.5.	REFERENCIAS	200

Capítulo 5. Relaciones entre clases

5.1.	TIPOS DE RELACIÓN ENTRE CLASES	205
	ASOCIACIÓN	206

DEPENDENCIA	209
GENERALIZACIÓN / ESPECIALIZACIÓN.....	209
AGREGACIÓN Y COMPOSICIÓN	211
REALIZACIÓN	212
PROBLEMA 18: SUMA DE DOS NÚMEROS	213
5.2. PAQUETES.....	219
PROBLEMA 19: VENTA DE PRODUCTOS.....	211
5.3. EJERCICIOS PROPUESTOS.....	227
5.4. REFERENCIAS	229
Capítulo 6. Mecanismos de herencia	
6.1. HERENCIA	235
HERENCIA SIMPLE	236
HERENCIA MÚLTIPLE: INTERFACES.....	238
PROBLEMA 20: EMPLEADOS POR HORA Y A DESTAJO	240
6.2. POLIMORFISMO.....	246
6.3. EJERCICIOS PROPUESTOS.....	248
6.4. REFERENCIAS	251
Apéndices	
A. ENTORNO INTEGRADO DE DESARROLLO SISMOO	252
B. ELEMENTOS SINTÁCTICOS DEL SEUDO LENGUAJE	260
C. GLOSARIO BÁSICO DE PROGRAMACIÓN ORIENTADA A OBJETOS.....	264
Índice analítico	271

Agradecimientos

Las sugerencias y aportes de estudiantes, egresados y profesores de la Facultad de Informática del Tecnológico de Antioquia fueron significativas durante la escritura de este libro. En particular, agradecemos a los estudiantes de Tecnología en Sistemas: Carlos Andrés García, Gloria Jazmín Hoyos, Steven Lotero, Emanuel Medina y Luis Gabriel Vanegas; a los Tecnólogos en Sistemas: Orlando Alarcón, Yeison Andrés Manco y Jader Rojas; a los ingenieros y profesores Eucario Parra, Gildardo Quintero, Darío Soto y Luis Emilio Velásquez; y al Comité para el Desarrollo de la Investigación del Tecnológico de Antioquia –CODEI–, en especial a la ex directora de Investigación y Posgrados, Amanda Toro, y a su actual director Jorge Ignacio Montoya.

Presentación

*Lógica y programación orientada a objetos: Un enfoque basado en problemas, es el fundamento teórico y práctico para un primer curso de programación. Además de incursionar de manera directa en el aprendizaje de un lenguaje que soporta el paradigma orientado a objetos¹, incluye ejercicios de aplicación de los conceptos propios del paradigma, a saber: clase, objeto, encapsulación, paquete y herencia, que conllevan a otros como atributo, método, visibilidad, constructor, estado de un objeto, recolector de basura, ligadura estática y ligadura dinámica. De esta manera, el libro responde a propósitos de renovación pedagógica orientados al diseño de currículos “con base en la investigación, que promueven la calidad de los procesos educativos y la permanencia de los estudiantes en el sistema”, uno de los desafíos de la educación en Colombia propuesto en el Plan Nacional Decenal de Educación vigente [PNDE2006]. También responde a la estructura curricular del módulo *Desarrollar Pensamiento Analítico Sistémico I* del proyecto Alianza Futuro Digital Medellín, del plan de estudios 72 del programa Tecnología en Sistemas del Tecnológico de Antioquia - Institución Universitaria.*

1 El paradigma orientado a objetos es un modelo o patrón para la construcción de software, entre otros existentes como los paradigmas imperativo, funcional y lógico.

Conviene señalar que este libro es el resultado del proyecto Sismoo – Sistema para el modelamiento por objetos –, adelantado en la línea de investigación “Ingeniería de software y sistemas de información”, inscrita en el grupo GIISTA – Grupo de Investigación en Ingeniería de Software del Tecnológico de Antioquia–, clasificado a 2009 en la categoría C de Colciencias. El proyecto Sismoo fue también una consecuencia de otro proyecto de la línea: MIPS00 – Método Integrado de Programación Secuencial y programación Orientada a Objetos para el análisis, diseño y elaboración de algoritmos–, que incluye un pseudo lenguaje y una didáctica para el aprendizaje de la programación. Ahora bien, el proyecto Sismoo aporta un intérprete del pseudo lenguaje propuesto por el MIPS00, sencillo y fácil de utilizar, que busca agilizar los procesos de enseñanza y aprendizaje de la programación de computadoras, e incluye un traductor al lenguaje de programación Java. Se proyecta seguir mejorando este objeto de aprendizaje.

El método para el aprendizaje y la enseñanza de la programación orientada a objetos del proyecto MIPS00 incluye elementos didácticos del *aprendizaje basado en problemas* y principios pedagógicos constructivistas. Su didáctica propone el seguimiento de cuatro fases para la solución de problemas:

- Definición de la tabla de requerimientos.
- Diseño del diagrama de clases.
- Definición de las responsabilidades de las clases.
- Desarrollo del pseudo código (tiene una gran similitud con los lenguajes de programación Java, C# y Visual Basic.Net)

Como complemento de MIPS00, Sismoo es un proyecto de investigación aplicada cuyo producto de desarrollo es una herramienta de software diseñada para que el usuario pueda editar, compilar y ejecutar los problemas diseñados empleando el pseudo lenguaje propuesto por MIPS00. Esta herramienta, denominada en el medio informático *Entorno Integrado de Desarrollo o IDE* (Integrated Development Environment), fue desarrollada en lenguaje Java y su diagrama estructural se asemeja al de cualquier compilador (ver Figura 1):

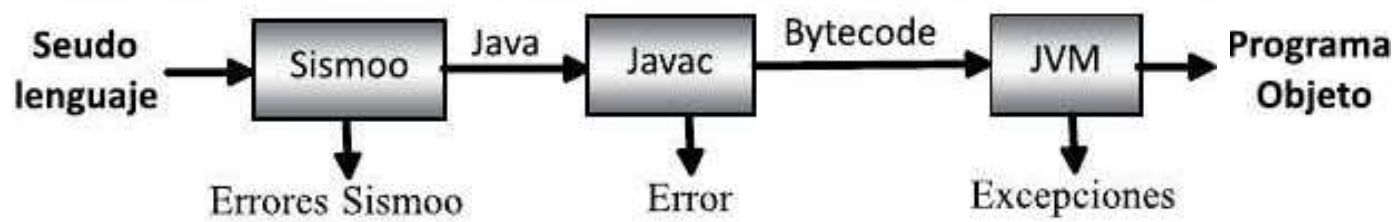


Figura 1. IDE traductor SISM00

Conocidos los objetivos y antecedentes de este libro, examinemos brevemente las características de sus lectores y capitulación.

Como todo texto académico, este libro presupone dos lectores principales: estudiantes y profesores, en particular, de tecnología e ingeniería de sistemas y áreas afines; también admite otros lectores con conocimientos básicos en programación estructurada o imperativa.

Los profesores encontrarán en el capítulo 1 (*Una didáctica para el aprendizaje de la programación*) y en el apéndice B (*Elementos sintácticos del pseudo lenguaje*), claves para el diseño de problemas propios de la programación; sin embargo se recomienda a los estudiantes la lectura de los apartados 1.2 y 1.3., donde pueden enterarse de las fases para la solución de los problemas y los tipos de problemas, que encontrará en este libro. Los capítulos restantes desarrollan teóricamente el paradigma orientado a objetos² y presentan la didáctica para el aprendizaje de los fundamentos de programación.

Continuando con la descripción de los capítulos, en el número 2, (*Fundamentos de Programación Orientada a Objetos*), se exponen los fundamentos de la programación orientada a objetos, por ello su lectura es primordial para el estudiante, quien adquiere las bases para abordar la lectura de los próximos capítulos. El capítulo 3, (*Clases: tipos de datos abstractos*), profundiza lo concerniente al manejo de métodos e introduce el empleo de las sentencias de control y los interruptores; aquí el estudio de la recursión es importante, en tanto puede retomarse para un curso de estructuras de datos. El capítulo 4, (*Arreglos*), es una incursión a las clases contenedoras lineales Vector y Matriz (clases que almacenan conjuntos de objetos); es también un abre bocas a un módulo

² Varios profesores nos referimos a él como “paradigma objetual”

o curso relacionado con estructuras de datos donde se puede profundizar mucho más. En los capítulos 5 (*Relaciones entre clases*) y 6 (*Mecanismos de herencia*) se tratan las relaciones fundamentales entre clases, como las asociaciones y dependencias, y los paquetes.

Una vez revisado el contenido del libro, quien tenga conocimientos del paradigma objetual, sea estudiante o profesor, y desee interactuar de entrada con un micro mundo de clases reutilizables (una visión bastante reducida de algo como la API de Java o los espacios de nombres de .Net Framework), puede comenzar con la lectura de los capítulos 2 y 3, y pasar a los capítulos finales, 5 y 6.

Se recomienda a un lector casual empezar a estudiar las historias que preceden cada capítulo, cuentos ilustrados y representados con un diagrama de clases, en consonancia con un artículo publicado por Zapata [Zapata1998].

Desde ahora es conveniente la lectura de los apéndices A (*Entorno Integrado de Desarrollo Sismoo*), donde se exponen los elementos necesarios para interactuar con el entorno integrado de desarrollo Sismoo; B (*Elementos sintácticos del pseudo lenguaje*) que presenta un listado de palabras reservadas para el pseudo lenguaje objetual, un resumen de las estructuras de control y los elementos estructurales que permiten definir una clase, un método, un paquete y una interfaz; y C (*Glosario básico de programación orientada a objetos*), que presenta un listado de términos con su significado para consulta rápida en el proceso de aprendizaje de la programación.

A la vez, durante la lectura de este libro, puede ser consultado su índice analítico, el cual contiene términos de relevancia que a nivel conceptual, deben ser claros para plantear soluciones concretas a los problemas sugeridos; en el libro, estos conceptos se escribieron en *cursiva* para facilitar su identificación en las páginas que señala el índice. El CD anexo incluye el intérprete Sismoo, además el JDK6.7 con uno de sus entornos (NetBeans 6.1) y el Microsoft .Net Framework 2.0 para el trabajo con el entorno SharpDevelop 2.2.1.

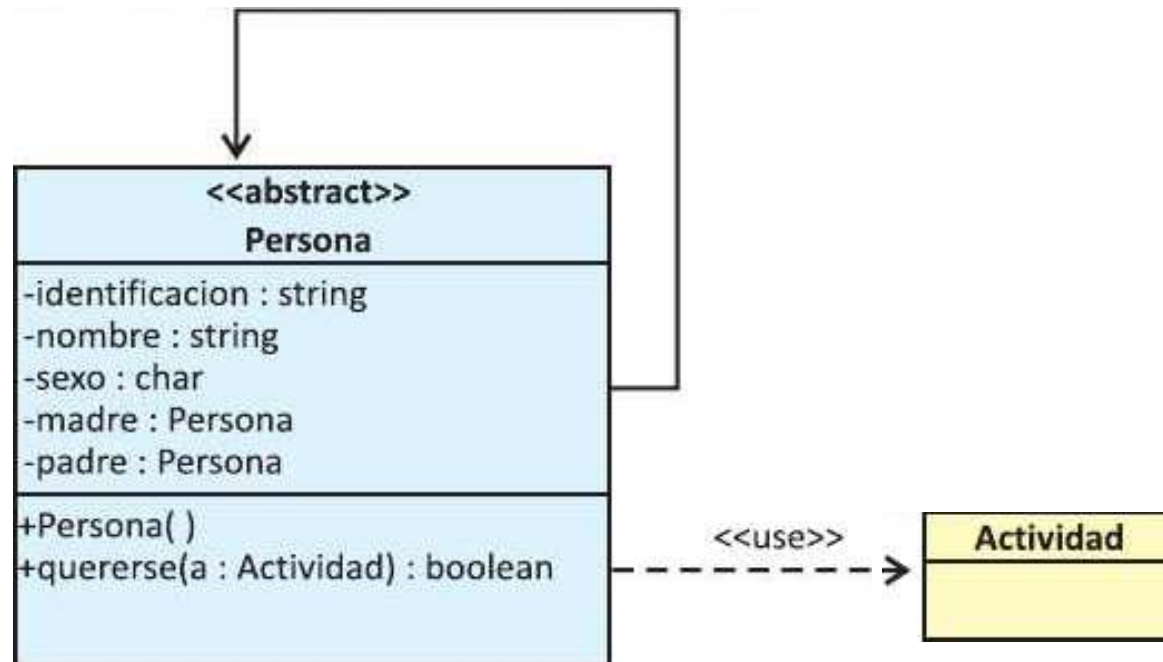
En la siguiente página el lector encontrará una frase célebre como pretexto a la familiarización con los diagramas de clase. Esperamos que este libro sea un disfrute para los estudiantes de programación y para sus profesores.

Los autores

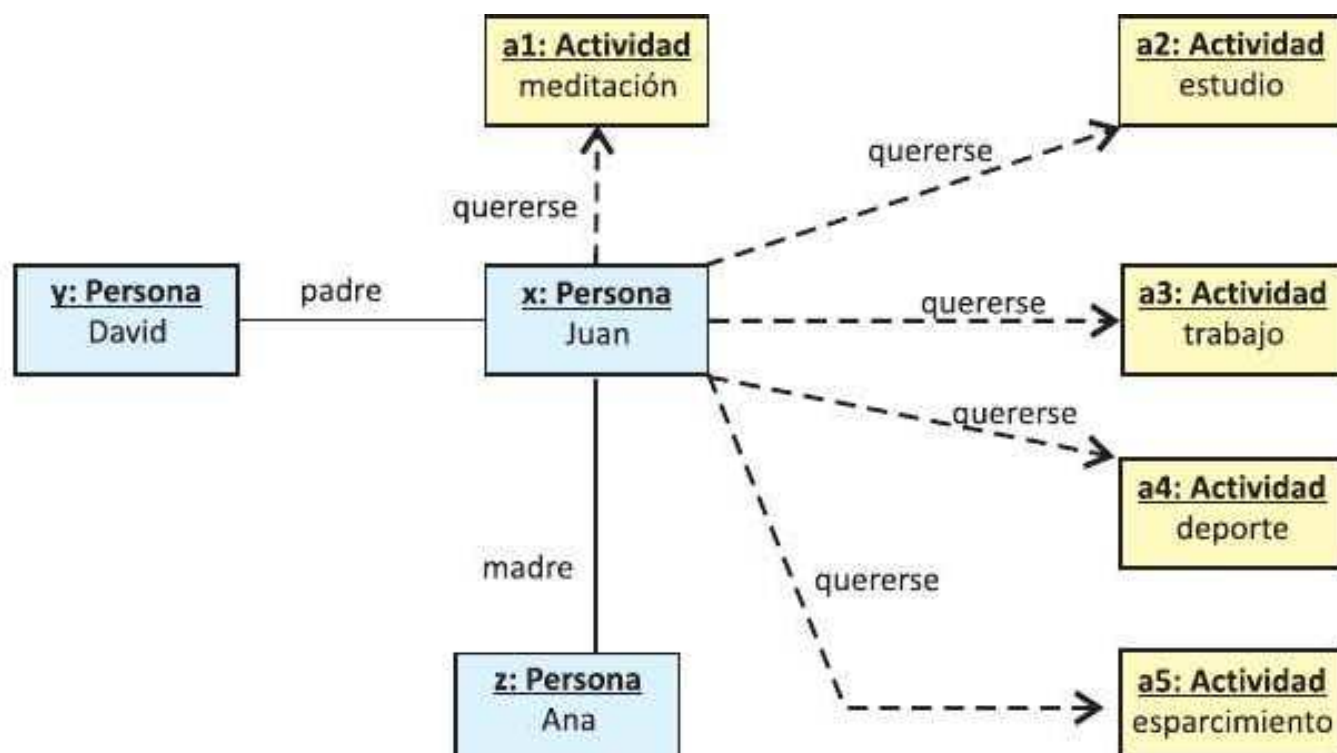
Quererse a sí mismo es el principio de un romance para toda la vida

Oscar Wilde

Diagrama de clases propuesto:



Un diagrama de objetos:



CAPÍTULO 1

Una didáctica para el aprendizaje de la programación

- 1.1. Algunas consideraciones pedagógicas para el aprendizaje de la programación
- 1.2. Fases para la solución de un problema
- 1.3. Tipo de problemas a tratar
- 1.4. Referencias

LA CLASE REAL

Real
- valor: real - const estatico real MIN_VALOR = $3.4 * 10^{-38}$ - const estatico real MAX_VALOR = $3.4 * 10^{38}$
+ Real () + asignarValor (real x) + obtenerValor (): real + obtenerMinValor(): real + obtenerMaxValor(): real + estatico obtenerMinValor(): real + estatico obtenerMaxValor(): real + esIgual (Real x): lógico + convBin (real x): Cadena + convBin (): Cadena + estatico convBin (real x): Cadena + convOctal (real x): Cadena + convOctal (): Cadena + estatico convOctal (real x): Cadena + convHex (real x): Cadena + convHex (): Cadena + estatico convHex (real x): Cadena + convertirANum (cadena x): real + estatico convertirANum (cadena x): real

Miembros dato de la clase Real:

valor: cantidad que representa el valor real, comprendido en el rango de $3.4 * 10^{-38}$ a $3.4 * 10^{38}$.

MIN_VALOR: constante estática real igual a $3.4 * 10^{-38}$

MAX_VALOR: constante estática real igual a $3.4 * 10^{38}$

Tabla 2.7. Métodos de la clase Real

MÉTODO	DESCRIPCIÓN
Real ()	Constructor por defecto. Inicializa el miembro dato <i>valor</i> en 0.
asignarValor (real x)	Asigna el real x al miembro dato <i>valor</i> .
obtenerValor (): real	Devuelve el valor actual del miembro dato <i>valor</i> .
obtenerMinValor (): real	Devuelve el mínimo valor real.
obtenerMaxValor (): real	Devuelve el máximo valor real.
estatico obtenerMinValor (): real	Devuelve el mínimo valor real. Método de clase.
estatico obtenerMaxValor (): real	Devuelve el máximo valor real. Método de clase.
esIgual (Real x): lógico	Devuelve cierto si el miembro dato <i>valor</i> es equivalente a x. En caso contrario devuelve falso.
convBin (real x): Cadena	Convierte en binario el real x. Retorna el resultado como un objeto de tipo Cadena.
convBin (): Cadena	Retorna una cadena igual al equivalente binario del miembro dato <i>valor</i> .
estático convBin (real x): Cadena	Retorna una cadena igual al equivalente binario del parámetro x. Método de clase.
convOctal (real x): Cadena	Convierte a base octal el real x. Retorna el resultado como un objeto de tipo Cadena.
convOctal (): Cadena	Retorna una cadena igual al equivalente octal del miembro dato <i>valor</i> .
estático convOctal (real x): Cadena	Retorna una cadena igual al equivalente octal del parámetro x. Método de clase.
convHex (real x): Cadena	Convierte a base hexadecimal el real x. Retorna el resultado como un objeto de tipo Cadena.
convHex (): Cadena	Retorna una cadena igual al equivalente hexadecimal del miembro dato <i>valor</i> .
estático convHex (real x): Cadena	Retorna una cadena igual al equivalente hexadecimal del parámetro x. Método de clase.
convertirANum(Cadena x): real	Convierte una cadena a número real.
estático convertirANum (cadena x): real	Convierte una cadena a número real. Método de clase.

LA CLASE CARACTER

Caracter
- valor: caracter - const estatico caracter MIN_VALOR = nulo - const estatico caracter MAX_VALOR = ' '
+ Carácter () + asignarValor (Caracter x) + obtenerValor (): Caracter + esDígito (Caracter x): logico + esDígito (): logico + esLetra (Caracter x): lógico + esLetra (): logico + esEspacio (Caracter x): logico + esEspacio (): logico + aMinúscula (Caracter x): caracter + aMinúscula (): caracter + aMayúscula (Caracter x): caracter + aMayúscula (): caracter + estatico convertirACarácter (cadena x): caracter

El miembro dato *valor* es una cantidad que representa un carácter, comprendido entre MIN_VALOR (nulo, carácter ASCII 0) y MAX_VALOR (blanco, carácter ASCII 255).

Tabla 2.8. Métodos de la clase Carácter

MÉTODO	DESCRIPCIÓN
Caracter ()	Constructor por defecto. Inicializa el miembro dato <i>valor</i> con un espacio en blanco.
asignarValor (Caracter x)	Modifica el estado del objeto, al asignar el dato x al miembro dato <i>valor</i> .
obtenerValor (): Caracter	Retorna el valor actual del atributo <i>valor</i> .
esDígito (Caracter x): logico	Devuelve cierto si el <i>argumento</i> x corresponde a un dígito. De lo contrario, falso.
esDígito (): logico	Devuelve cierto si el miembro dato <i>valor</i> corresponde a un dígito. De lo contrario, falso.

esLetra (caracter x): logico	Devuelve cierto si el <i>argumento x</i> corresponde a un Caracter alfabético. De lo contrario, falso.
esLetra (): logico	Devuelve cierto si el miembro dato <i>valor</i> corresponde a un carácter alfabético. De lo contrario, falso.
esEspacio (caracter x): logico	Devuelve cierto si el argumento <i>x</i> corresponde a un espacio en blanco. De lo contrario devuelve falso.
esEspacio (): logico	Devuelve cierto si el miembro dato <i>valor</i> corresponde a un espacio en blanco. De lo contrario devuelve falso.
aMinúscula (Caracter x): caracter	Devuelve la letra minúscula correspondiente al carácter <i>x</i> .
aMinúscula (): caracter	Devuelve la letra minúscula correspondiente al miembro dato <i>valor</i> .
aMayúscula (caracter x): caracter	Devuelve la letra mayúscula correspondiente al carácter <i>x</i> .
aMayúscula (): caracter	Devuelve la letra mayúscula correspondiente al miembro dato <i>valor</i> .
convertirACarácter (cadena x): caracter	Convierte una cadena en carácter. Si la cadena consta de varios caracteres, se retorna el primer carácter de la cadena.

LA CLASE LOGICO

Logico
- valor: logico
+ Logico () + asignarValor (logico x) + obtenerValor (): logico + iguales (logico x): logico

La clase cadena será expuesta como una clase de uso común.

Nota: Se sugiere al docente y a los estudiantes desarrollar algunos de los métodos de las clases base de uso común antes especificadas, a manera de ejercicios para el repaso y aplicación de los conceptos estudiados.

PAQUETES DE USO COMÚN

La solución de problemas en diferentes contextos, permite identificar que un grupo de clases se utiliza en todas o en la mayoría de las soluciones planteadas. Estas clases se denominan de uso común y pueden empaquetarse para efectos de reutilización. Esta característica también las agrupa en *paquetes* sistema y contenedor. El primero de ellos (sistema) se importa por defecto e incluye las clases que encapsulan los tipos de datos primitivos (Entero, Real, Caracter, Logico, Cadena). Mientras que la clase Mat, cuya utilidad radica en las funciones matemáticas de uso frecuente, contiene la clase Flujo para controlar la entrada y salida estándar, la clase Cadena para el tratamiento de cadenas de caracteres y la clase Excepcion para el control de los errores en tiempo de ejecución. El segundo paquete (contenedor) incluye tipos abstractos de datos contenedores denominados *estructuras de datos internas* (residen en memoria RAM), como las clases Arreglo, Vector, Matriz, Pila, Cola, Listaligada, Árbol, ListaOrdenada, ABB (árbol binario de búsqueda) y Grafo; la única *estructura de datos externa* de este paquete (reside en memoria externa como el disco duro de la computadora o una memoria USB) es la clase Archivo. Los siguientes diagramas representan los *paquetes de uso común*:

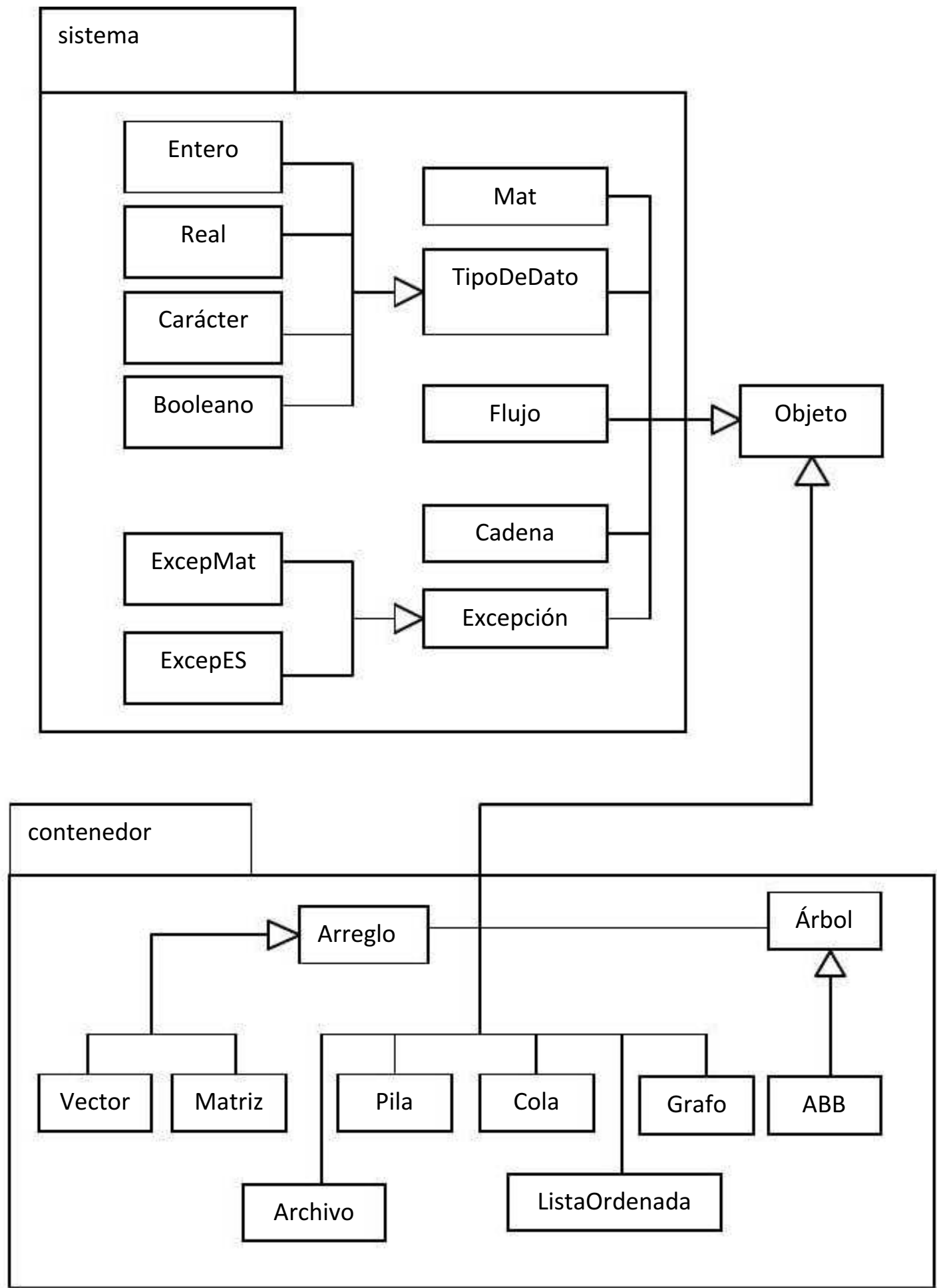


Figura 2.7. Paquetes de uso común

Observación: todas las clases contenidas en los paquetes de uso común se desligan de la *clase Objeto*, siendo ésta la clase superior de la jerarquía.

2.5. CONFIGURACIÓN DE LAS CLASES DE USO COMÚN

Cada clase de uso común se configura con los atributos y operaciones que le caracterizan, lo que permite solucionar ciertos problemas de manera rápida con simples mecanismos de reutilización.

A continuación se presenta la configuración de las clases Objeto, TipoDeDato, Flujo, Cadena y Mat, con la correspondiente descripción de sus atributos.

LA CLASE OBJETO

La *clase Objeto* es la raíz de todo el árbol de la jerarquía de clases uso común, y proporciona cierto número de métodos de utilidad general que pueden utilizar todos los objetos.

Objeto
+ comparar (Objeto x): entero + aCadena (): cadena + aEntero(): entero + aReal(): real + aCaracter(): caracter + obtenerClase ()

Métodos de la clase:

Tabla 2.8. Métodos de la clase Objeto

MÉTODO	DESCRIPCIÓN
comparar(Objeto x) : entero	Compara objetos entre sí. Esta comparación no es la misma que proporciona el operador ==, que solamente se cerciora si dos referencias a objetos apuntan al mismo objeto El método <i>comparar()</i> se usa para saber si dos objetos separados son del mismo tipo y contienen los mismos datos. El método devuelve un valor entero que puede ser: <ul style="list-style-type: none">• igual a cero, si los objetos son iguales• menor que cero, si el objeto actual es menor que el objeto x• mayor que cero, si el objeto actual es mayor que el objeto x
aCadena() : cadena	Convierte un objeto en una cadena. Aquí, el método aCadena() extrae el entero contenido en un objeto Entero y lo retorna como una cadena
aEntero() : entero	Convierte un objeto en un número entero
aReal() : real	Convierte un objeto en un número real
aCaracter() : caracter	Convierte un objeto en un carácter
obtenerClase()	Devuelve la clase de un objeto. Este método se utiliza para determinar la clase de un objeto, es decir, devuelve un objeto de tipo Clase, que contiene información importante sobre el objeto que crea la clase. Una vez determinada la clase del objeto, se pueden utilizar sus métodos para obtener información acerca del objeto

LA CLASE TIPODEDATO

TipoDeDato es una clase abstracta utilizada para definir, mediante los mecanismos de herencia, las clases asociadas a los tipos de datos estándar.

TipoDeDato <<abstracta>>
+ asignarValor() + obtenerValor()

Por polimorfismo, las clases Entero, Real, Caracter y Logico deben implementar los métodos asignarValor () y obtenerValor (). Si una clase es abstracta, sus métodos también lo son.

LA CLASE FLUJO

La *clase Flujo* ofrece los servicios de ingreso de datos desde el dispositivo estándar de entrada (teclado) y visualización de información en el dispositivo estándar de salida (monitor).

Flujo
+ estatico leerCadena(): cadena + estatico leerEntero(): entero + estatico leerReal(): real + estatico leerCaracter(): Caracter + estatico leerObjeto(): Objeto + estatico imprimir(<lista_de_argumentos>)

Tabla 2.9. Métodos de la clase Flujo

MÉTODO	DESCRIPCIÓN
leerCadena() : cadena	Lee una cadena de caracteres desde el teclado.
leerEntero() : entero	Lee un número entero.
leerReal() : real	Lee un número real.
leerCaracter() : caracter	Lee un carácter.
leerObjeto() : objeto	Lee un objeto. Este método puede reemplazar a cualquiera de los anteriores.
imprimir()	Visualiza en pantalla el contenido de los argumentos. La lista de argumentos puede ser de cualquier tipo primitivo, objetos, variables o constantes numéricas y de cadena, separadas por el símbolo de concatenación cruz (+).

Algunos ejemplos de instrucciones de lectura y escritura donde se utiliza la clase Flujo, se presentan en el siguiente segmento de pseudo código:

```
entero cod
real precio
cadena nombre

Flujo.imprimir ("Ingrese el código, nombre y precio del artículo:")
cod = Flujo.leerEntero( )
precio = Flujo.leerReal( )
nombre = Flujo.leerCadena( )
Flujo.imprimir ("codigo: " + cod + "Nombre: " + nombre + "Precio: $" + precio)
```

Obsérvese el uso de los métodos estáticos de la clase Flujo: imprimir(), leerEntero(), leerReal() y leerCadena().

LA CLASE CADENA

La *clase Cadena* contiene las operaciones básicas que permiten administrar cadenas de caracteres en un algoritmo.

Cadena
- cad []: caracter - longitud: entero - const estatico entero MAX_LONG = 256
+ Cadena () + asignarCad (Cadena x) + obtenerCad (): Cadena + car (entero x): caracter + num (carácter x): entero + valor (Cadena x): entero + valor (Cadena x): real + convCad (entero x): Cadena + convCad (real x): Cadena + estatico comparar (Cadena c1, Cadena c2): entero

La clase Cadena contiene tres miembros dato:

cad: arreglo unidimensional de tipo caracter.

longitud: cantidad de caracteres de la cadena (equivale al número de elementos o tamaño del vector *cad*).

MAX_LONG: es una constante entera y estática que determina la longitud máxima de la cadena, dada por 256 caracteres.

Descripción de los métodos:

Tabla 2.10. Métodos de la clase Cadena

MÉTODO	DESCRIPCIÓN
Cadena ()	Constructor que inicializa el atributo <i>cad</i> con la cadena vacía.
asignarCad (Cadena x)	Asigna el objeto <i>x</i> al objeto mensajero.
obtenerCad(): Cadena	Retorna el contenido del miembro privado <i>cad</i> .
car (entero x): caracter	Devuelve el código ASCII correspondiente al entero <i>x</i> .
num (caracter x): entero	Devuelve el número entero asociado a un carácter ASCII.
valor (Cadena x): entero	Convierte una cadena en un valor numérico entero.
convCad (entero x): Cadena	Convierte un entero en una cadena.
comparar(Cadena c1, Cadena c2): entero	Compara la cadena <i>c1</i> con la cadena <i>c2</i> . Devuelve 0 si las cadenas son iguales, -1 si $c1 < c2$ y 1 si $c1 > c2$.

LA CLASE MAT

Esta clase, carente de atributos, contiene una serie de métodos estáticos (funciones matemáticas) susceptibles de invocar desde cualquier algoritmo, sin la necesidad de pasar un mensaje a un objeto determinado.

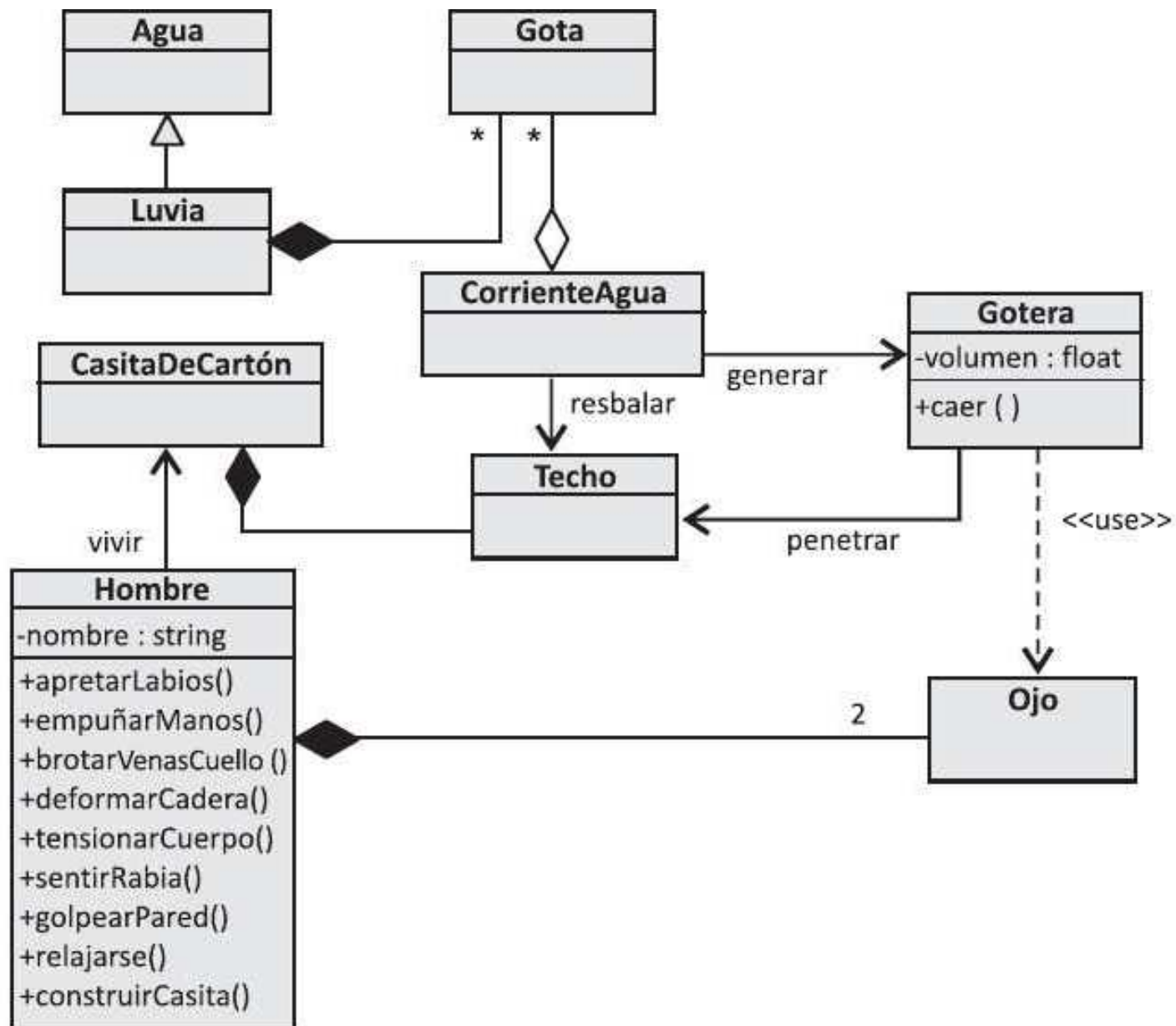
RABIA

Diego Aristizábal, 2007

Esa noche, por culpa de una gotera que le caía en un ojo, el hombre se despertó con los labios apretados, con las manos empuñadas, con las venas brotadas en el cuello, con la cadera deformada, con la voz iracunda insultándose por dentro, con el cuerpo tensionado y con tanta rabia que pensó que golpearía por primera vez una pared, pero no lo hizo. Si lo hacía, tendría que relajarse al instante para construir de nuevo su casita de cartón.



Diagrama de clases:



Objetivos de aprendizaje

- Familiarizarse con las estructuras de control selectivas y repetitivas.
- Plantear soluciones con métodos estáticos o métodos de clase.
- Construir la tabla de requerimientos, el diagrama de clases, las responsabilidades de las clases (expresadas con los contratos de los métodos) y el seudo código, para una serie de problemas de carácter convencional.
- Resolver problemas utilizando métodos recursivos.

3.1. ESTRUCTURA GENERAL DE UNA CLASE

Como se ha notado en los cuatro problemas tratados hasta el momento, una *clase* se puede definir en seudo código o de forma gráfica utilizando los formalismos de UML. La representación en seudo código para una *clase* incluye algunos elementos excluyentes entre sí (la visibilidad de la clase, por ejemplo) y otros opcionales (visibilidad, herencia e interfaz), como se observa a continuación:

```
[abstracto | final] [público | privado | protegido] clase nomClase [heredaDe
                                nomClaseBase] [implementa nomInterfaz]

    // Cuerpo de la clase
fin_clase
```

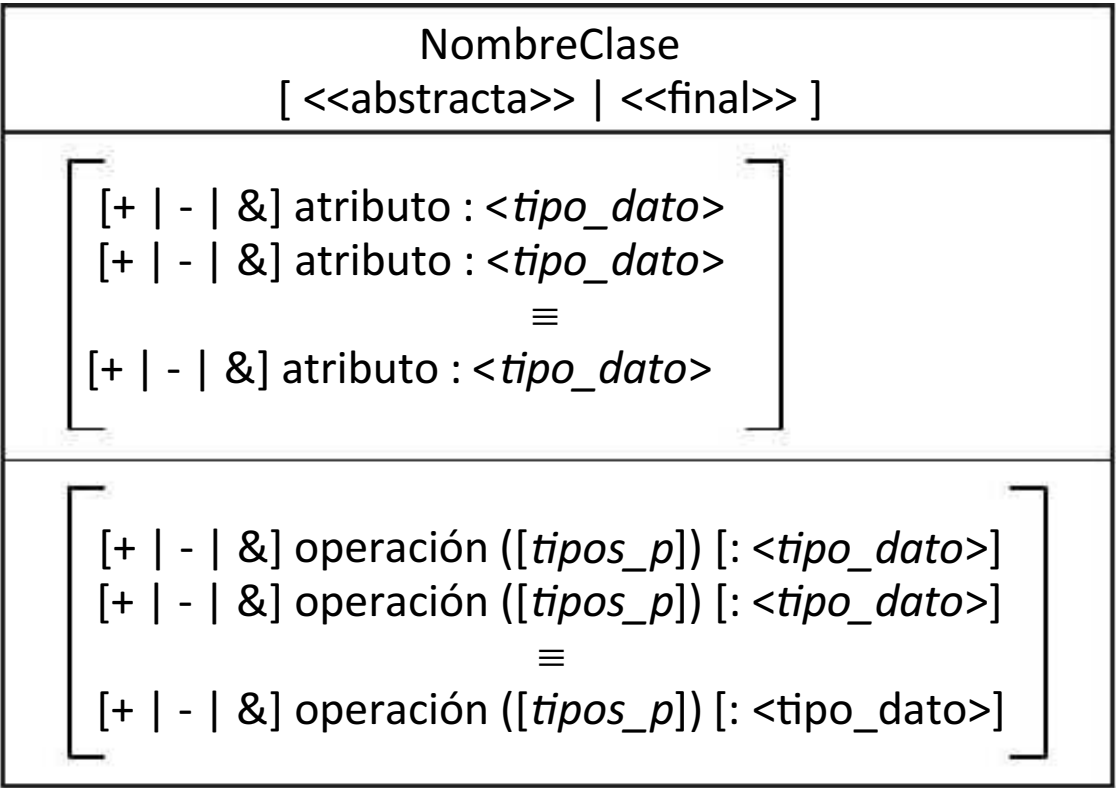
El cuerpo de la clase incluye dos secciones: una dedicada a los *atributos* y otra dedicada a las *operaciones*. Ambas partes son opcionales, lo que permite, en términos de software, definir una clase sin atributos o sin operaciones. Esta característica no concuerda con la realidad, pues tanto atributos como operaciones son inherentes a la naturaleza de las clases.

El *cuerpo de una clase* tiene la siguiente configuración:

```
[público | privado | protegido]:
    atributo : <tipo_dato>
    atributo : <tipo_dato>
    ≡
    atributo : <tipo_dato>

[público | privado | protegido]:
    operación ([tipos_p]) [: <tipo_dato>]
    operación ([tipos_p]) [: <tipo_dato>]
    ≡
    operación ([tipos_p]) [: <tipo_dato>]
```

De igual manera y para efectos de un curso introductorio de fundamentos de programación, la estructura parcial de una clase en UML es la siguiente:



Observaciones:

- Para ambos casos -representación en pseudo código y en UML-, lo que se encierra entre corchetes es opcional. El carácter barra vertical (|) significa que se debe escoger una opción entre las disponibles, por tanto, una clase se declara pública, privada o protegida, mientras que una *clase abstracta* nunca puede ser final.

- El orden en la declaración de atributos y operaciones no interesa. Sin embargo, según la sintaxis de UML, se deben especificar primero los atributos y luego las operaciones.
- Cada elemento sintáctico propone estos significados:

público, privado, protegido: hace referencia al *especificador de acceso, alcance o visibilidad* de la clase, atributo u operación.

La visibilidad según UML se representa con los signos + (público), - (privado) y & (protegido).

Si no se indica la visibilidad, se asume pública.

abstracta: denota a una *clase abstracta*, aquella desde la cual no se pueden instanciar objetos y que permite definir otras clases que heredan de ella.

final: indica que la clase finaliza la jerarquía, es decir, no se pueden definir clases derivadas a partir de ella.

clase: palabra reservada que permite definir una nueva clase.

- NombreClase: es el nombre de un *tipo abstracto de datos*. Por convención el nombre de una clase inicia con una letra mayúscula y las restantes con minúsculas. Para el caso de un nombre de clase compuesto por varias palabras, se puede optar por el uso de guión como separador de palabras o concatenar todo el identificador comenzando toda nueva palabra con una letra mayúscula. Ejemplos: Empleado, EmpleadoOficial, Empleado_oficial
- atributo: nombre válido de un atributo de clase. Siempre comienza con una letra, puede incluir dígitos, letras y carácter de subrayado, no admite espacios y nunca puede coincidir con una palabra reservada.
- <tipo_dato>: corresponde a un tipo de datos estándar o a un tipo abstracto de datos.

Se tendrán en cuenta cinco tipos de datos estándar: entero, real, carácter, cadena y lógico. Todo atributo siempre debe tener un tipo asociado. Sin embargo, en una operación se puede omitir el tipo si ésta no retorna valor.

- *operación()*: corresponde a una función o procedimiento que realiza algún tipo de operación con los atributos de la clase. Las *operaciones* son los algoritmos que se deben desarrollar para darle funcionalidad a la clase.
- *tipos_p*: lista de tipos de datos correspondiente a los *parámetros*. Aquí se debe observar que no interesa el nombre del *argumento*, pero sí el tipo. Los nombres de los argumentos cobran relevancia al momento de definir la operación.
- Si las operaciones de una clase se definieran en línea, estilo lenguaje Java, la expresión [+ | - | &] operación ([argumentos]) : [<tipo_dato>], se conoce como *declaración del método* o *definición de prototipo*.

3.2. MÉTODOS

Un *método* es un pequeño programa diseñado para efectuar determinada tarea en la solución de un problema. En el problema 2, se identificaron los métodos *asignarX()*, *obtenerY()*, *dibujar()*, y *principal()*, este último de carácter inamovible y obligatorio para toda solución planteada, pues constituye el punto de entrada y salida de la aplicación.

El término *subprograma* es propio del paradigma de programación imperativo o estructurado, y es comúnmente utilizado cuando hay independencia entre los datos y los algoritmos que los tratan. En programación orientada a objetos, el término subprograma es reemplazado por *método*, donde los datos y los algoritmos se integran en un componente denominado *objeto*.

Los subprogramas, y por tanto los métodos, se clasifican en procedimientos y funciones. Un *procedimiento* puede o no tener parámetros y puede retornar cero, uno o

más valores al método que lo invocó. Una *función* tiene cero, uno o más parámetros y siempre retorna un valor.

DEFINICIÓN DE UN MÉTODO

La definición de un método incluye la especificación de *visibilidad*, *tipo de retorno*, *estereotipado*, *nombre del método*, *parámetros*, *cuerpo* y *finalización*.

```
[público | privado | protegido] [<tipo_dato>] [estático]
    nombre_método ([lista_parámetros])
        // Instrucciones o cuerpo del método
    [retornar(valor)]
fin_método
```

Observaciones:

- Lo que se encuentra encerrado entre corchetes es opcional. El carácter barra vertical o pipe (|) indica que se debe seleccionar entre una de las opciones; así, una clase es pública, privada o protegida.
- Las palabras reservadas público, privado y protegido hacen referencia al *especificador de acceso* o *visibilidad* de la operación. Si no se indica la visibilidad se asume que es pública.
- <tipo_dato> corresponde al *tipo de retorno* (tipo del resultado devuelto por el método), el cual puede corresponder con un tipo de datos estándar o un tipo abstracto de datos.

Se tendrán en cuenta los cinco tipos de datos estándar: entero, real, carácter, cadena y lógico. Si no se especifica el tipo de retorno, se asume vacío.

- La palabra *estático* indica que el método así lo es.
- *nombre_método* es cualquier identificador válido para el nombre de un método, con las siguientes restricciones:

- No puede coincidir con una palabra reservada (ver el listado de las mismas en el apéndice B, ítem B1).
- Debe comenzar con una letra o con el carácter de subrayado. No se admiten comienzos con dígitos o caracteres especiales.
- No puede contener espacios.
- La cantidad de caracteres del identificador es irrelevante, aunque se aconseja el uso de identificadores con una longitud que no exceda los 15 caracteres.
- El nombre del método debe conservar la mnemotecnica adecuada, es decir, recordar en términos generales la tarea que realiza.

Algunos identificadores inválidos son:

3rValor (comienza con un dígito), para (coincide con una palabra reservada del pseudo lenguaje), #TotalPctos (comienza con un carácter especial no permitido), cuenta corriente (incluye un espacio).

Por el contrario, los siguientes identificadores son correctos:

valorFinal, PI, direccionOficina, promedio_anual, mes7.

- Una operación que emule a una función debe incluir la sentencia retornar. Si la operación señala a un procedimiento, no se incluye dicha sentencia.
- *lista_parámetros* es una lista de variables u objetos separados por comas, con sus tipos respectivos. Los *parámetros* permiten la transferencia de datos entre un objeto perteneciente a la clase y el mundo exterior; forman parte de la interfaz pública de la clase. Una de las condiciones para aquella transferencia de datos es que en el momento del *paso de un mensaje* al objeto exista una relación uno a uno con los *argumentos* en cuanto a orden, número y tipo.
- El *cuerpo del método* constituye su desarrollo algorítmico; puede incluir declaración de variables, constantes, objetos, registros, tipos enumerados y sentencias de control para toma de decisiones e iteración de instrucciones.
- La instrucción retornar (*valor*) es opcional. Si el método tiene algún tipo de retorno, la variable local *valor* debe ser del mismo tipo que *< tipo_dato >*, es decir, el tipo de retorno del método.

- `fin_método` es una palabra reservada que indica la *finalización del método*; la palabra *método* se puede escribir sin tilde.

INVOCACIÓN DE UN MÉTODO

La *invocación de un método* implica adaptarse a la siguiente sintaxis:

```
[variable = ] nombre_objeto.nombre_método([lista_parámetros_actuales])
```

Observaciones:

- Si el método retorna un valor, este se debe asignar a una variable.
- Se debe tener en cuenta que la lista de parámetros actuales debe coincidir con la lista de *argumentos* dada en la declaración de prototipo de la clase.

Ejemplo:

Suponga que la función *factorial()* y el procedimiento *mostrarVector()* están definidos como métodos no estáticos de la clase *Arreglo* como se indica a continuación:

```
entero factorial(entero n)
    entero f = 1, c
    para (c = n, c > 1, c = c - 1)
        f = f * c
    fin_para
    retornar f
fin_método
// -----
vacío mostrarVector(real vec[ ])
    entero i = 0
    mientras (i < vec.longitud)
        Flujo.imprimir(vec[i])
        i = i + 1
    fin_mientras
fin_método
// -----
```

Posibles invocaciones para cada uno de estos métodos serían:

- Invocación dentro de otro método de la clase *Arreglo*:

```
// Factorial de 7 y de un entero ingresado por el usuario
entero x7 = factorial(7)
entero n = Flujo.leerEntero( )
Flujo.imprimir("El factorial de 7" + " es " + x7)
Flujo.imprimir("El factorial de " + n + " es " + factorial(n))

// ...

real valores[ ] = {1.2, 5.7, -2.3, 4.0, 17.1, 2.71}
mostrarVector(valores)
```

- Invocación dentro de un método que no pertenece a la clase *Arreglo*:

```
//Creación de un objeto de la clase Arreglos
Arreglo x = nuevo Arreglo ( )
Flujo.imprimir("El factorial de 7 es " + x.factorial(7))
real valores[ ] = {1.2, 5.7, -2.3, 4.0, 17.1, 2.71}
Flujo.imprimir("Las " + valores.longitud + " constantes reales son:")
x.mostrarVector(valores)
```

MÉTODOS ESTÁTICOS

Un *método estático*, denominado también *método de clase*, no requiere de una instancia de objeto para ejecutar la acción que desarrolla, puesto que tiene la semántica de una función global.

La sintaxis general para invocar a un método estático es la siguiente:

NombreDeClase.métodoEstático([parámetros])

asignarNombre (cadena x)	R1	El empleado tiene un nombre igual a la cadena vacía	El empleado tiene un nombre dado por el parámetro x	Asignar al atributo <i>nombre</i> el valor de x
obtenerNombre(): cadena	R1 R2	El usuario desconoce el nombre del empleado	El usuario conoce el nombre del empleado	Devolver el valor del atributo <i>nombre</i>
asignarSalario (real x)	R1	El empleado tiene un salario igual a cero	El empleado tiene un salario dado por el parámetro x	Asignar al atributo <i>salario</i> el valor de x
obtenerSalario(): real	R1 R2	El usuario desconoce el salario del empleado	El usuario conoce el salario del empleado	Devolver el valor del atributo <i>salario</i>


* R. a.: Requerimiento asociado

d) Seudo código

```

clase Empleado
  privado:
    cadena nombre
    real salario
  publico:
    Empleado( )
    fin_metodo
    //-----
    Empleado(cadena n, real s)
        nombre = n
        salario = s
    fin_metodo
    //-----
    asignarNombre(cadena x)
        nombre = x
    fin_metodo
    //-----
    cadena obtenerNombre( )
        retornar nombre
    fin_metodo
    //-----
    asignarSalario(real x)
        salario = x
    fin_metodo
    //-----
    real obtenerSalario( )
        retornar salario
    fin_metodo
    //-----
    estatico cadena nombreMayorSalario( Empleado a, Empleado b)
        cadena nom
        si (a.obtenerSalario( ) > b.obtenerSalario( ))
            nom = a.obtenerNombre( )
        sino
            si (b.obtenerSalario( ) > a.obtenerSalario( ))
                nom = b.obtenerNombre( )
            sino
                nom = " "
            fin_si
        fin-si
        retornar nom
    fin_metodo
fin_clase
//*****

```





```

clase Proyecto
    publico principal ( )
1      cadena nom
2      Empleado e1 = nuevo Empleado( )
3      Empleado e2 = nuevo Empleado( )
4
5      Flujo.imprimir("Ingrese nombre y salario de dos empleados:")
6      e1.asignarNombre(Flujo.leerCadena( ))
7      e1.asignarSalario(Flujo.leerReal( ))
8      e2.asignarNombre(Flujo.leerCadena( ))
9      e2.asignarSalario(Flujo.leerReal( ))
10     nom = Empleado. nombreMayorSalario(e1, e2)
11     si (Cadena.comparar(nom, " "))
12         Flujo.imprimir("Ambos empleados devengan igual salario")
13     sino
14         Flujo.imprimir("Quien más devenga entre " +
                        e1.obtenerNombre( ) + " y " + e2.obtenerNombre( ) +
                        " es " + nom)
15     fin_si
16
17     Empleado e3 = nuevo Empleado("Adán Picasso", 4500000)
18     Empleado e4 = nuevo Empleado("Eva Modigliani", 2000000)
19
20     nom = Empleado. nombreMayorSalario(e3, e4)
21     si (Cadena.comparar(nom, " "))
22         Flujo.imprimir("Ambos empleados devengan igual salario")
23     sino
24         Flujo.imprimir("Quien más devenga entre " +
                        e3.obtenerNombre( ) + " y " + e4.obtenerNombre( ) +
                        " es " + nom)
25     fin_si
    fin_metodo
fin_clase

```

Observaciones:

- El método *nombreMayorSalario()* es estático y retorna el nombre del empleado que devenga el mayor salario. Dado el caso que ambos empleados obtengan un igual salario, el método retorna una cadena vacía.

- El cuerpo del método *principal()* se ha enumerado para facilitar ciertas explicaciones. Las líneas 4, 16 y 19 están vacías: cuestión de estilo en la presentación del pseudo lenguaje. Las instrucciones 14 y 24 conllevan tres líneas de pseudo código.
- El método constructor fue sobrecargado una vez, agregando dos parámetros: uno de tipo cadena para el nombre y otro de tipo real para el salario. Esto permite crear objetos de dos formas distintas, tal como se observa en las líneas 2, 3, 17 y 18.

PROBLEMA 7: INFORME SOBRECARGADO

Determinado usuario, del cual se conoce su nombre, desea obtener los valores correspondientes al seno y coseno de dos ángulos establecidos por él mismo, uno de tipo entero y otro de tipo real. Se pide visualizar dos informes: el primero debe incluir un saludo previo y el nombre del usuario. El saludo depende del tiempo del día (mañana, tarde o noche) y puede ser uno de tres: “Buen día”, “Buena tarde” o “Buena noche”. El segundo informe debe mostrar los valores de las funciones solicitadas y el mayor entre los dos ángulos ingresados.

Véanse un par de salidas posibles:

- *Buen día Sr(a) Ava Gardner.*

Para 30 y 52.8 grados, tenemos:

$$\text{Seno}(30) = 0.50 \qquad \text{Coseno}(30) = 0.87$$

$$\text{Seno}(52.8) = 0.80 \qquad \text{Coseno}(52.8) = 0.60$$

El ángulo mayor es 52.8.

- “Buena tarde Sr(a) Juan D’Arienzo.

Para 45 y 7.5 grados, tenemos:

$$\text{Seno}(45) = 0.71 \qquad \text{Coseno}(45) = 0.71$$

$$\text{Seno}(7.5) = 0.13 \qquad \text{Coseno}(7.5) = 0.99$$

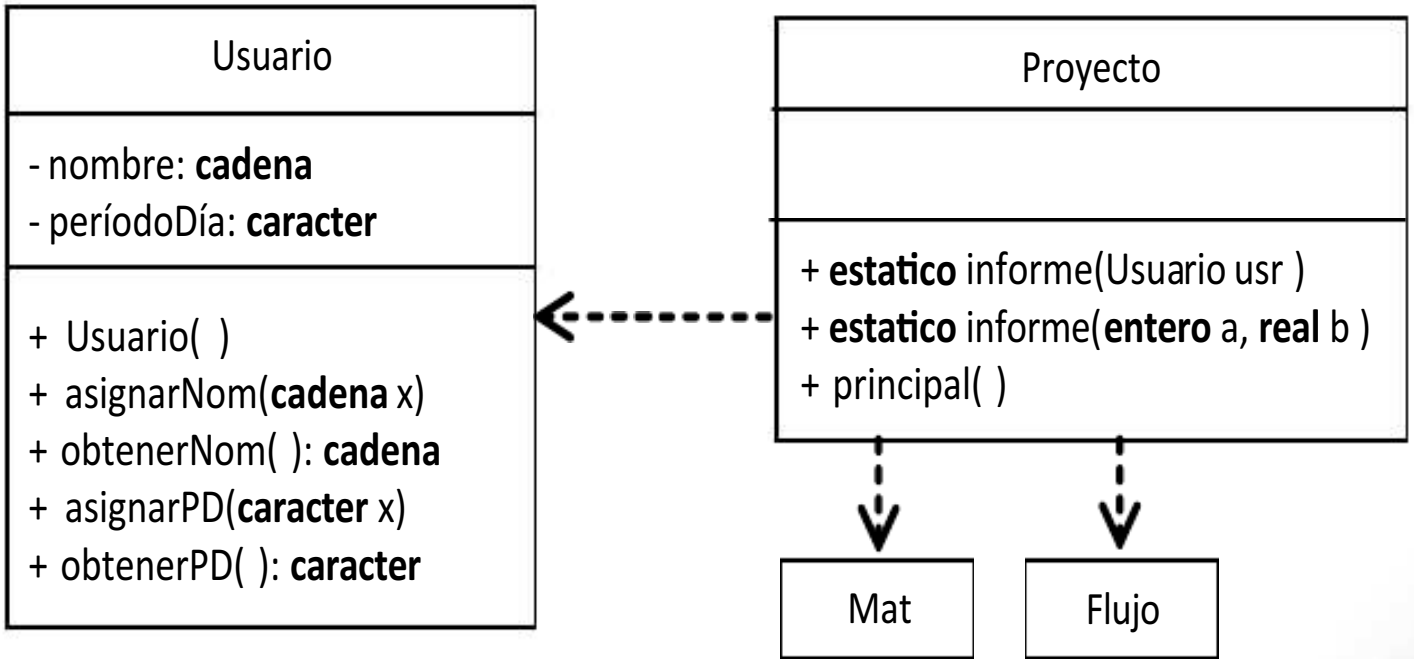
El ángulo mayor es 45.

Solución:

a) Tabla de requerimientos

Identificación del requerimiento	Descripción	Entradas	Resultados (Salidas)
R1	Conocer el nombre del usuario y el tiempo del día	Los valores digitados por el usuario	Nombre de usuario y período del día establecidos
R2	Visualizar el primer informe, que incluye un saludo y el nombre del usuario	Nombre del usuario y período del día ('m': Mañana, 't': Tarde, 'n': Noche)	Primer informe emitido
R3	Visualizar el segundo informe, que incluye seno y coseno de los dos ángulos y el mayor de los ángulos	Los valores de dos ángulos	Segundo informe emitido

b) Diagrama de clases



Observaciones:

- El método *informe()* está sobrecargado una vez: la definición inicial del método acepta un objeto tipo *Usuario* como parámetro, con el objetivo de elaborar el informe inicial. El método sobrecargado acepta como parámetros los valores entero y real de los dos ángulos, en búsqueda de elaborar el informe final.
- Para el cálculo de las funciones trigonométricas seno y coseno se utilizarán los métodos estáticos *Mat.sen()* y *Mat.cos()*, ambos sobrecargados para recibir argumentos enteros y reales, como se observa en la sesión 2.5 (Configuración de las clases de uso común).

c) Responsabilidades de las clases
Contrato de la clase Usuario

Método	R. a.*	Precondición	Postcondición
Usuario()	R1	Ninguna	Existe un objeto tipo <i>Usuario</i> en memoria
asignarNom(cadena x)	R1	Un objeto tipo <i>Usuario</i>	El atributo <i>nombre</i> del objeto tipo <i>Usuario</i> ha sido asignado
obtenerNom(): cadena	R1	Se desconoce el nombre del usuario	El nombre del usuario se ha obtenido
asignarPD(caracter x)	R1	Un objeto tipo <i>Usuario</i>	El atributo <i>períodoDía</i> del objeto tipo <i>Usuario</i> ha sido asignado
obtenerPD(): caracter	R1	Se desconoce el período del día en el cual el usuario requiere los informes	El período del día se ha obtenido

* Requerimiento asociado

Contrato de la clase Proyecto

Método	R. a.	Precondición	Postcondición
estatico informe(Usuario u)	R2	Conocer los datos de un usuario	Se ha visualizado el primer informe
estatico informe(entero a, real b)	R3	Conocer los dos ángulos	Se ha visualizado el segundo informe
principal()	R1 R2 R3	Conocer los datos de un usuario y los valores de los ángulos	Se han visualizado los dos informes

Observación:

La columna “*Modelo verbal*” se ha omitido en los contratos de las dos clases, porque los métodos son realmente simples. Esta columna se expone cuando los métodos presentan algún grado de complejidad o es muy extenso el proceso lógico de los mismos. Queda al libre elección del analista incluir o no esta columna dentro del contrato de las clases.

d) Seudo código

Clase Usuario

privado:

cadena nombre

carácter períodoDía

publico:

Usuario()

fin_metodo


//-----

asignarNom(cadena x)

nombre = x

fin_metodo

//-----



```

↓
    obtenerNom( ): cadena
        retornar nombre
    fin_metodo
//-----
    asignarPD(caracter x)
        períodoDía = x
    fin_metodo
//-----
    obtenerPD( ): carácter
        retornar períodoDía
    fin_metodo
//-----
fin_clase
//*****
Clase Proyecto
    publico:
        estatico informe(Usuario usr )
            según (usr.obtenerPD( ))
                caso 'm':
                    Flujo.imprimir("Buen día")
                    saltar
                caso 't':
                    Flujo.imprimir("Buena tarde")
                    saltar
                caso 'n':
                    Flujo.imprimir("Buena noche")
            fin_según
            Flujo.imprimir(" Sr(a) " + usr.obtenerNom ( ))
        fin_metodo
//-----
        estatico informe(entero a, real b )
            Flujo.imprimir("Para " + a + " y " + b + " grados, tenemos:")
            Flujo.imprimir("Seno(" + a + ") =" + Mat.sen(a))
            Flujo.imprimir("Coseno(" + a + ") =" + Mat.cos(a))
            Flujo.imprimir("Seno(" + b + ") =" + Mat.sen(b))
            Flujo.imprimir("Coseno(" + b + ") =" + Mat.cos(b))
            Flujo.imprimir("El ángulo mayor es ")
            si (a > b)
                Flujo.imprimir(a)
            sino
                Flujo.imprimir(b)
            fin_si
        fin_metodo
//-----
↓

```



```

principal( )
    Usuario u = nuevo Usuario( )
    entero a1
    real a2
    Flujo.imprimir("Ingrese nombre del usuario: ")
    u.asignarNom(Flujo.leerCadena( ))
    Flujo.imprimir("Ingrese período del día ('m': mañana, `t`: tarde, `n`: noche): ")
    u.asignarPD(Flujo.leerCaracter( ))
    informe(u)
    Flujo.imprimir("Ingrese el valor de los dos ángulos: ")
    a1 = Flujo.leerEntero( )
    a2 = Flujo.leerReal( )
    informe(a1, a2)
fin_metodo
//-----
fin_clase

```

3.4. ESTRUCTURAS DE CONTROL

Las *estructuras de control*, como su nombre lo indica, permiten controlar el flujo de ejecución de un método; tienen una entrada y una salida y se clasifican en tres tipos:

- Secuencia: ejecución sucesiva de una o más instrucciones.
- Selección: ejecuta uno de dos posibles conjuntos de instrucciones, dependiendo de una condición (expresión lógica) o del valor de una variable.
- Iteración: repetición de una o varias instrucciones a la vez que cumple una condición.

LA ESTRUCTURA SECUENCIA

En la *estructura secuencia*, las instrucciones se aparecen una a continuación de la otra, en secuencia lineal, sin cambios de ruta. Una sintaxis general de la estructura secuencia está dada por:

<i>instrucción_1</i> <i>instrucción_2</i> : : <i>instrucción_N</i>
--

Ejemplo:

```
x = Flujo.leerEntero()  
r = Mat.seno(x) - 7.5  
Flujo.imprimir (r)
```

LA ESTRUCTURA SELECCIÓN

La *estructura selección* admite dos variantes: la *decisión* y el *selector múltiple*. La primera, identificada con la *sentencia* *si*, evalúa una condición y, dependiendo de su valor de verdad, ejecuta un bloque de instrucciones u otro. La segunda, identificada con la *sentencia* *según*, evalúa una variable denominada “selectora” y ejecuta uno de tres o más casos.

Sintaxis de la instrucción de decisión:

<i>si</i> (<i>e</i>) <i>instrucciones_1</i> [<i>sino</i> <i>instrucciones_2</i>] <i>fin_si</i>
--

Donde,

e: expresión lógica.

instrucciones_1: bloque de instrucciones a ejecutar si la expresión *e* es verdadera.

instrucciones_2: bloque de instrucciones a ejecutar si *expresión* es falsa.

La cláusula sino es opcional, por eso va encerrada entre corchetes. Una *cláusula* es una palabra reservada que forma parte de una instrucción; en este caso, la instrucción de decisión incluye la sentencia si y dos cláusulas: sino y fin_si.

Ejemplos de estructuras de decisión:

— Una instrucción por bloque

```
si (x % 2 == 0)
    Flujo.imprimir(x + "es par")
sino
    Flujo.imprimir(x + "es impar")
fin_si
```

— Sin cláusula sino

```
si (num > 0)
    valor = num * 2
fin_si
```

— Varias instrucciones por bloque

```
si (x % 2 == 0)
    Flujo.imprimir(x + "es par")
    sw = cierto
sino
    Flujo.imprimir(x + "es impar")
    sw = falso
    cont = cont + 1
fin_si
```

- Condición compuesta

```
    si (a > b && a > c)
        Flujo.imprimir("El mayor es " + a)
    fin_si
```
- *Decisión anidada*: bloque decisivo dentro de otro

```
    si (nombre.comparar("John Neper"))
        indicativo = cierto
        c1 = c1 + 1
    sino
        si (nombre.comparar("Blaise Pascal"))
            indicativo = falso
            c2 = c2 + 1
        fin_si
    fin_si
```

No existe un límite para el número de anidamientos.

El *selector múltiple* tiene la siguiente sintaxis:

```
según (vs)
    caso c1:
        instrucciones_1
    caso c2:
        instrucciones_2
    caso cN:
        instrucciones_N
    [sino
        instrucciones_x]
fin_según
```

Donde,

vs: variable selectora. Debe ser de tipo entero, carácter, tipo enumerado o subrango.

3.7. EJERCICIOS PROPUESTOS

Métodos

1. Escriba métodos que permitan:
 - Hallar el dato mayor entre tres elementos.
 - Visualizar la media, mediana y desviación típica para un conjunto de datos.
 - Imprimir el valor total de la serie:

$$(1 - x) + (3 + x)^2 + (5 - x)^4 + (7 + x)^8 + \dots$$
 El método recibe como parámetros el valor de x y la cantidad de términos a generar.

Sentencias de control

2. Solucionar el problema nueve (Estadísticas por procedencia), utilizando otra clase además de la que contiene el método *principal* (). El enunciado de dicho problema es el siguiente:
 Para un número determinado de personas se conoce su estatura, procedencia y edad. La estatura y la procedencia se manejan de acuerdo a las siguientes convenciones:

Estatura = 1 (alta), 2 (baja) o 3 (Mediana)

Procedencia = 'L' (Americana), 'E' (Europea), 'A' (Asiática) u 'O' (Otra).

Determinar:

- El número de americanos altos, europeos bajos y asiáticos medianos.
 - La edad promedio de los individuos de otras procedencias.
 - La cantidad de americanos bajos mayores de edad.
3. Hallar los datos mayor y menor entre una muestra de n números enteros ingresados por el usuario.
 Presentar dos estilos de solución: una con aplicación de sentencias de control según lo explicado en este capítulo, otra con aplicación de las clases de uso común, según el capítulo 2.

Recursión

4. Sean b y p números enteros mayores o iguales a cero. Calcular b^p : el número b elevado a la potencia p .
5. El máximo común divisor de los enteros x y y es el mayor entero que divide tanto a x como a y . La expresión $x \% y$ produce el residuo de x cuando se divide entre y . Defina el máximo común divisor (mcd) para n enteros x y y mediante:

$$\text{mcd}(x, y) = y \text{ si } (y \leq x \ \&\& \ x \% y == 0)$$

$$\text{mcd}(x, y) = \text{mcd}(y, x) \text{ si } (x < y)$$

$$\text{mcd}(x, y) = \text{mcd}(y, x \% y) \text{ en otro caso.}$$

Ejemplos.: $\text{mcd}(8, 12) = 4$, $\text{mcd}(9, 18) = 9$, $\text{mcd}(16, 25) = 1$, $\text{mcd}(6, 15) = 3$.

6. Suponga que $\text{com}(n, k)$ representa la cantidad de diferentes comités de k personas que pueden formarse, dadas n personas entre las cuales elegir. Por ejemplo $\text{com}(4, 3) = 4$, porque dadas cuatro personas A, B, C y D hay cuatro comités de tres personas posibles: ABC, ABD, ACD y BCD.

Para m valores, compruebe la identidad

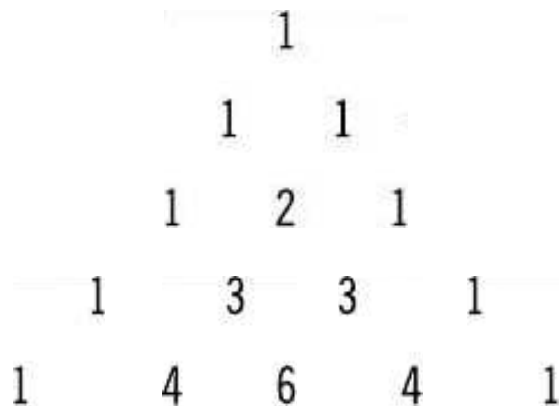
$$\text{com}(n, k) = \text{com}(n - 1, k) + \text{com}(n - 1, k - 1), \text{ donde } n, k \geq 1.$$

7. Los coeficientes binomiales pueden definirse por la siguiente relación de recurrencia, que es el fundamento del triángulo de Pascal:

$$c(n, 0) = 1 \text{ y } c(n, n) = 1 \text{ para } n \geq 0$$

$$c(n, k) = c(n - 1, k) + c(n - 1, k - 1) \text{ para } n > k > 0.$$

Ejemplo: triángulo de Pascal de altura 5.



Defina $c(n, k)$ para dos enteros cualquiera $n \geq 0$ y $k > 0$, donde $n \geq k$.

8. Multiplicar n pares de números naturales por medio de la recurrencia

$$a * b = \begin{cases} a & \text{si } b == 1 \\ a * (b - 1) + a & \text{si } b > 1 \end{cases}$$

donde a y b son enteros positivos.

9. La función de Ackerman se define para los enteros no negativos m y n de la siguiente manera:

$$\begin{aligned} a(m, n) &= n + 1 \text{ si } m = 0. \\ a(m, n) &= a(m - 1, 1) \text{ si } m \neq 0, n == 0. \\ a(m, n) &= a(m - 1, a(m, n - 1)) \text{ si } m \neq 0, n \neq 0. \end{aligned}$$

Compruebe que $a(2, 2) = 7$.

10. El siguiente es un algoritmo recursivo para invertir una palabra:

```

si (la palabra tiene una sola letra)
    Esta no se invierte, solamente se escribe.
sino
    Quitar la primera letra de la palabra.
    Invertir las letras restantes
    Agregar la letra quitada.
fin_si
  
```

Por ejemplo, si la palabra a invertir es ROMA, el resultado será AMOR; si la palabra es ANILINA (cadena capicúa), el resultado será ANILINA.

Invertir n palabras, donde la variable n debe ser especificada por el usuario.

3.8. REFERENCIAS

[Aristizábal2007]: Aristizábal, Diego Alejandro (2007). Un Cuento para tu Ciudad en Cien Palabras. Metro de Medellín, Tercer puesto.

[Bobadila2003]: Bobadilla, Jesús. Java a través de ejemplos. Alfaomega-Rama, Madrid, 2003.

[Eckel2002]: Eckel, Bruce. Piensa en Java. Segunda edición, Pearson Educación, Madrid, 2002.

[Oviedo2004]: Oviedo, Efraín. Lógica de Programación. Segunda Edición. Ecoe Ediciones, Bogotá, 2004.

CAPÍTULO 4

Arreglos

- 4.1. Operaciones con arreglos
 - Declaración de un arreglo
 - Asignación de datos a un arreglo
 - Acceso a los elementos de un arreglo
 - Problema 14: Sucesión numérica almacenada en un vector
- 4.2. La clase Vector
 - Problema 15: Unión de dos vectores
 - Problema 16: Búsqueda binaria recursiva
- 4.3. La clase Matriz
 - Problema 17: Proceso electoral
- 4.4. Ejercicios propuestos
- 4.5. Referencias

ESE INOLVIDABLE CAFECITO

Juan Carlos Vásquez

Un día en el Instituto nos invitaron -a los que quisiéramos acudir-, a pintar una pobre construcción que hacía de colegio y que era el centro de un poblado de chozas, cuyo nombre no puedo acordarme, en una zona muy marginal, muy pobre y muy apartada de nuestras urbanizaciones, aunque, no muy distante.

Voluntariamente, acudió todo el curso, acompañado de nuestros hermanos guías, los promotores de la iniciativa solidaria.

Fue un sábado muy temprano, cuando montados en nuestras dos cafeteras de autobuses todos tan contentos, armados con nuestras respectivas brochas, para pintar de alegría y de esperanza, los rostros de aquella desconocida gente.

Cuando llegamos, vimos como unas veinte chozas alrededor de una pobre construcción de cemento que hacía de colegio y, escuchamos la soledad escondida, excluida, perdida.

Nos pusimos manos a la obra: unos arriba, otros abajo; unos dentro, otros fuera. Como éramos como ochenta pintores de brocha grande, la obra duró tan solo unas tres o cuatro horas.

Pero, antes de terminar, nos llamaron para que descansáramos, y salimos para fuera y vimos una humilde señora que nos invitaba a tomar café. La señora, con toda la amabilidad, dulzura, y agradecimiento, nos fue sirviendo en unas tacitas de lata que íbamos pasando a otros después de consumirlo.

Nunca olvidaré ese olor y ese sabor de café, pues quedó grabado en mi memoria olfativa y gustativa para siempre. Nunca me han brindado un café tan rico como el que nos ofrecieron en ese día solidario.

Fue un café dado con todo el amor del mundo. Me supo a humanidad, me supo a gloria.

Fue mi mejor café, el café más rico del mundo.

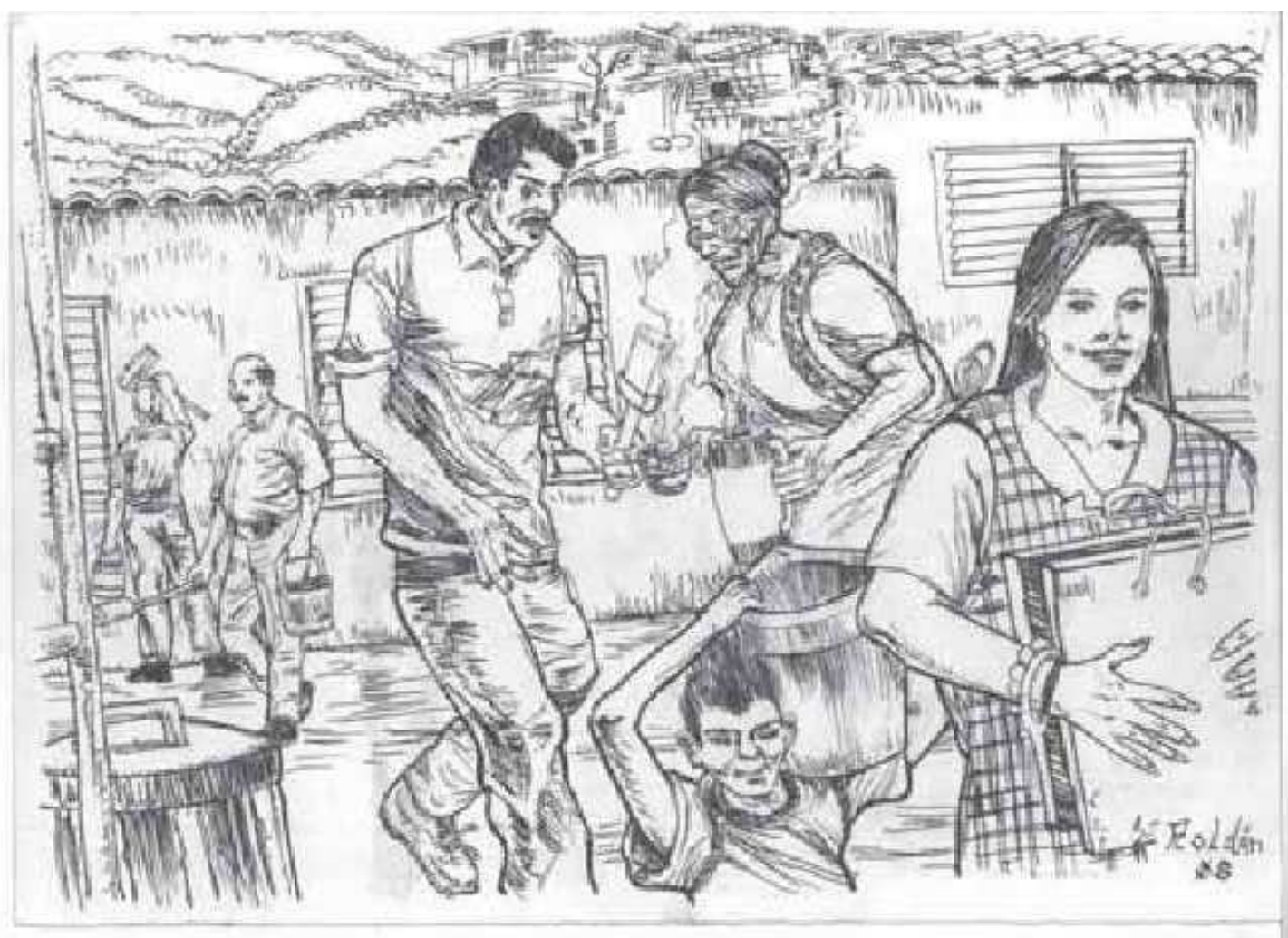
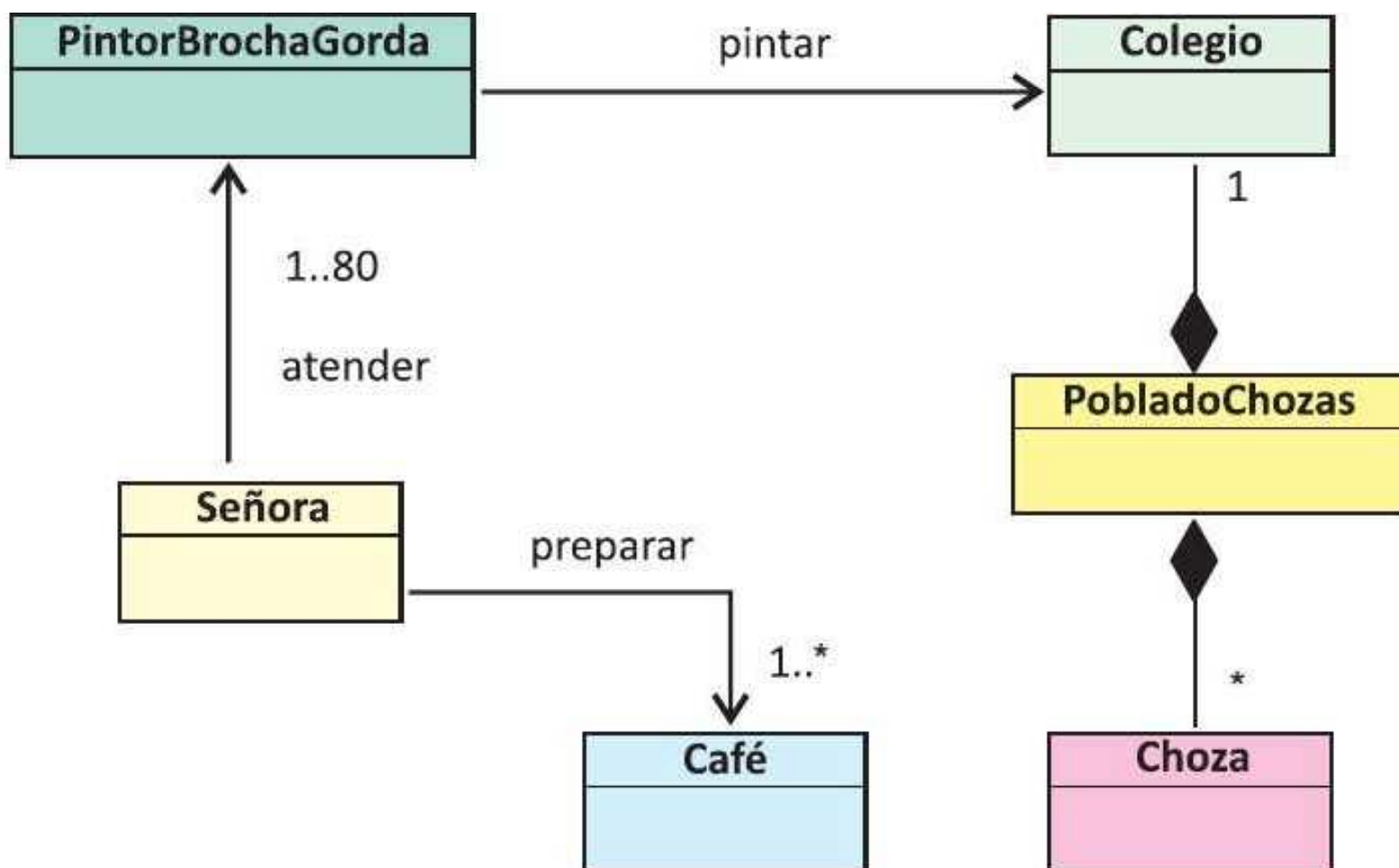


Diagrama de clases:



Objetivos de aprendizaje

- Resolver problemas mediante la utilización de las estructuras de datos contenedoras lineales Vector y Matriz.
- Manejar arreglos unidimensionales y bidimensionales de tipos estándar y tipos abstractos de datos.

Un *arreglo* es una *estructura de datos lineal y estática*, con nombre único, compuesta por elementos del mismo tipo; se considera lineal porque para cada elemento del arreglo, siempre existirán un predecesor o elemento anterior y un sucesor o elemento siguiente (exceptuando los elementos primero y último del arreglo), dando la sensación abstracta de linealidad en la disposición de los elementos que componen la estructura; se considera estática porque la memoria de computadora (RAM – Random Access Memory) que ocupará la estructura se debe definir en tiempo de compilación y no en tiempo de ejecución, como ocurre con las estructuras dinámicas como las listas ligadas.⁷ Esto significa que el programador debe conocer la máxima cantidad de elementos del arreglo para poder dimensionarlo antes de proceder con la ejecución del programa.

4.1. OPERACIONES CON ARREGLOS

Los arreglos se clasifican de acuerdo a su dimensión así:

- Unidimensionales o vectores.
- Bidimensionales o matrices.

⁷ Las estructuras de datos dinámicas como las listas enlazadas, así como la implementación dinámica de otras estructuras contenedoras de datos como las pilas, colas, árboles y grafos, forman parte de un segundo curso de lógica de programación. Para profundizar estas estructuras remitirse a las referencias [Cairó 1993], [Weiss1995] y [Flórez 2005].

- N-dimensionales, donde $N > 2$

Los arreglos pueden contener muchos elementos del mismo tipo, a modo de paquete de datos que se puede referenciar a través de un solo nombre; por esto se dice que un arreglo es una *estructura de datos contenedora*. Debe tenerse en cuenta que todo arreglo se debe declarar y dimensionar antes de comenzar a utilizarlo.

DECLARACIÓN DE UN ARREGLO

En general, la *declaración de un arreglo* tiene la siguiente sintaxis:

$\underbrace{\text{<tipo_dato> nomArreglo[] [[]... []]}_{\text{Declaración}} = \underbrace{\text{nuevo <tipo_dato> [d1] [[d2]... [dn]]}_{\text{Dimensionamiento}},$

Donde:

El miembro izquierdo de la asignación corresponde a la declaración del arreglo, y se explicita el tipo de datos del arreglo, su nombre y cantidad de dimensiones (cantidad de pares de corchetes vacíos); la parte derecha hace referencia al *dimensionamiento del arreglo*, es decir, al espacio de memoria que utilizará el arreglo.

<tipo_dato>: Es el tipo de datos del arreglo. Puede ser un tipo estándar (entero, real, caracter, logico, cadena), un tipo abstracto de datos (Producto, Cuenta, Planta, etc.) o un tipo estándar clasificado (Entero, Real, Caracter, Logico, Cadena).

nomArreglo: Es el nombre dado al arreglo, que debe cumplir todas las características para la construcción de identificadores.

[] [[]... []]: Son pares de corchetes, donde cada par indica una nueva dimensión del arreglo; del segundo al último son opcionales.

Donde:

- La clase Vector forma parte del paquete de uso común contenedor, presentado en la sesión 2.4 (*Paquetes de uso común*). Este paquete contiene otras clases (Matriz, ListaOrdenada, Pila, Cola, ABB, Grafo y Archivo) conocidas como *estructuras de datos internas* debido a que su objetivo es optimizar el manejo de grandes cantidades de datos en la memoria interna de la computadora. En este libro nos ocuparemos de las clases Vector y Matriz.
- La *clase* Vector posee tres atributos: un arreglo unidimensional de visibilidad privada denominado *vec*, que guarda datos de tipo Objeto; una variable *n* de visibilidad pública que almacena el número de datos que guarda el vector; y una variable *tam*, también de visibilidad pública, que contiene la máxima cantidad de elementos que puede almacenar el vector. Tener en cuenta el rango del atributo *n*: $0 \leq n \leq tam$.

Los atributos *n* y *tam* son públicos porque usualmente se utilizan en las diferentes operaciones que pueden hacerse sobre un vector: recorrido, búsqueda de un elemento, ordenamiento, entre otras.

- Esta clase maneja dos constructores. El primero, *Vector(entero x)*, tiene como parámetro el número de elementos que va a guardar el vector. Al tamaño *tam* se le suma 10 con el objetivo de conservar espacio de memoria adicional para futuras adiciones de elementos; este valor es subjetivo y puede variar de acuerdo a la conveniencia del caso. Este constructor inicializa en nulo todos los elementos del arreglo, como se detalla a continuación:

```
Vector (entero x)
entero i
    n = x
    tam = x + 10
    vec = nuevo Objeto[x]
    para (i = 0, i < x, i = i + 1)
        vec[i] = nuevo Objeto( )
        vec[i] = nulo
    fin_para
fin_metodo
```


Para crear un vector v de tamaño cien se escribe:

```
Vector v = nuevo Vector(100)
```

El segundo constructor, `Vector(Objeto v[])`, tiene como parámetro un arreglo unidimensional de tipo Objeto, e inicializa el atributo `vec` con los valores que guardan los elementos del parámetro `v` por medio de una única y simple operación de asignación, como se observa a continuación:

```
Vector(Objeto v[ ])
vec = v
n = v.n
    tam = v.tam
fin_metodo
```

Para crear un vector $v1$ que guarde los datos del arreglo $v2$ se escribe:

```
real v2[ ] = {1, 2, 3, 4}
Vector v1 = nuevo Vector(v1)
```

- Los métodos restantes se describen así:
 - `asignarDato(Objeto d, entero p)` → Asigna el dato d a la posición p del vector `vec`. Este método permite modificar el estado del vector porque el contenido de `vec[p]` cambia por d .
 - `obtenerDato(entero p): Objeto` → Retorna el contenido de `vec[p]`, es decir, permite conocer el estado del vector en su posición p .
 - `llenar()` → Llena el vector con n datos de tipo Objeto.
 - `mostrar()` → Imprime el contenido del vector `vec`.
 - `buscar(Objeto x): entero` → Busca el dato x en el vector `vec` y retorna la posición (índice del vector) donde fue hallado. Si x no existe retorna un valor igual a -1.

- insertar(Objeto x): lógico → Inserta el dato x al final del vector vec, es decir, en la posición $n + 1$. Si la inserción tuvo éxito se retorna cierto; si no hay espacio disponible porque $tam == n$, se retorna falso.
- eliminar(Objeto x): lógico → Elimina el dato x en el vector vec. Retorna cierto si x fue eliminado o falso en caso contrario.
- ordenar() → Ordena el vector vec de menor a mayor.

El pseudocódigo para la *clase Vector* es el siguiente:

```

clase Vector
  privado Objeto vec[ ]
  publico entero n
  publico entero tam

  publico:
    Vector(entero x)
      entero i
      n = x
      tam = x
      vec = nuevo Objeto[x]
      para (i = 0, i < x, i = i + 1)
        vec[i] = nulo
      fin_para
    fin_metodo
    //-----
    Vector(Objeto v[ ])
      vec = v
      n = v.n
      tam = v.tam
    fin_metodo
    //-----
    asignarDato (Objeto d, entero p)
      vec[p] = d
    fin_metodo
    //-----
    Objeto obtenerDato (entero p)
      retornar vec[p]
    fin_metodo
    //-----

```



```

llenar( )
    entero i
    para (i = 0, i < n, i = i + 1)
        vec[i] = Flujo.leerObjeto( )
    fin_para
fin_metodo
//-----
cadena mostrar( )
    cadena texto = " "
    entero i
    para( i = 0, i < n, i = i + 1)
        texto = texto + vec[i].aCadena( ) + " "
    fin_para
    retornar texto
fin_metodo
//-----
entero buscar(Objeto x)
    entero pos = 0 // Contador para recorrer el vector
    lógico hallado = falso // Supuesto: x no existe
    // Mientras no encuentre a x y existan
    elementos por comparar
    mientras (hallado == falso && pos < n)
        si (vec[pos].comparar(x) == 0) // ¿Lo encontró?
            hallado = cierto
        sino
            pos = pos + 1 // Avance a la siguiente posición
    fin_si
fin_mientras
si (hallado == falso) //¿No encontró a x en vec?
    pos = -1
fin_si
retornar pos
fin_metodo
//-----
lógico insertar(Objeto x)
    lógico indicativo = falso // No se ha insertado
    entero p = buscar(x) // Verifica si el dato a
    insertar existe en vec
    si (p == -1) // ¿No encontró el dato x?
        si (n < tam) // ¿Hay espacio disponible en el
            vector?
                vec[n] = x
                n = n + 1
                indicativo = cierto // Inserción realizada
            Flujo.imprimir("Inserción realizada")

```

```

↓
    sino
        Flujo.imprimir("No hay espacio para la inserción")
    fin_si
    sino
        Flujo.imprimir(x.aCadena( ) + " ya existe")
    fin_si
    retornar indicativo
fin_metodo
//-----
entero eliminar(Objeto x)
    entero p = buscar(x) // Verifica si el dato a eliminar existe en vec
    si (p != -1) ¿Encontró a x en la posición p?
        // Desde p, desplazar los elementos de vec una posición hacia atrás
        para (i = p, i < n - 1, i = i + 1)
            vec[i] = vec[i + 1]
        fin_para
        n = n - 1 // Borrado lógico de la última posición
        Flujo.imprimir (x.aCadena( ) + " eliminado")
    sino
        Flujo.imprimir(x.aCadena( ) + " no existe")
    fin_si
    retornar p
fin_metodo
//-----
ordenar( )
    entero i, j
    Objeto aux
    para (i = 0, i < n - 1, i = i + 1)
        para (j = i + 1, j < n, j = j + 1)
            si (vec[i].comparar(vec[j]) > 0)
                aux = vec[i]
                vec[i] = vec[j]
                vec[j] = aux
            fin_si
        fin_para
    fin_para
fin_metodo
//-----
fin_clase

```

Observaciones:

- El método *mostrar()* utiliza la variable local *texto* de tipo cadena para crear una representación textual del vector a imprimir. Cada elemento del vector *vec* es convertido a texto mediante el método *aCadena()* de la clase *Objeto*, con el fin de concatenarlo a la variable *texto* a través el operador cruz (+).

A manera de ejemplo, para mostrar el vector *v* en pantalla, basta con escribir las instrucciones:

```
real v1[] = {1.7, 2.8, 3.9}
Vector v = nuevo Vector(v1)
v.mostrar()
```

- El método *buscar()* devuelve la posición -número entero- donde fue hallado el dato *x*. En el proceso de búsqueda del elemento, se utiliza el método *comparar()* de la clase *Objeto*.
- Los métodos *insertar()* y *eliminar()* verifican si el dato *x* existe o no dentro del vector *vec*, porque este contiene claves de búsqueda -no puede contener elementos repetidos-. En los ejercicios propuestos del capítulo se plantea un problema sobre un vector que puede contener elementos duplicados.
- El método *insertar()* inserta el dato *x* al final del vector *vec*, siempre y cuando exista espacio disponible.
- La eliminación de un elemento del vector implica un borrado lógico en lugar de un borrado físico. En la figura 4.2. se observa el proceso que se lleva a cabo cuando se desea eliminar el dato *x*, ubicado en la posición *pos* del vector *vec*, que en este caso almacena números enteros.

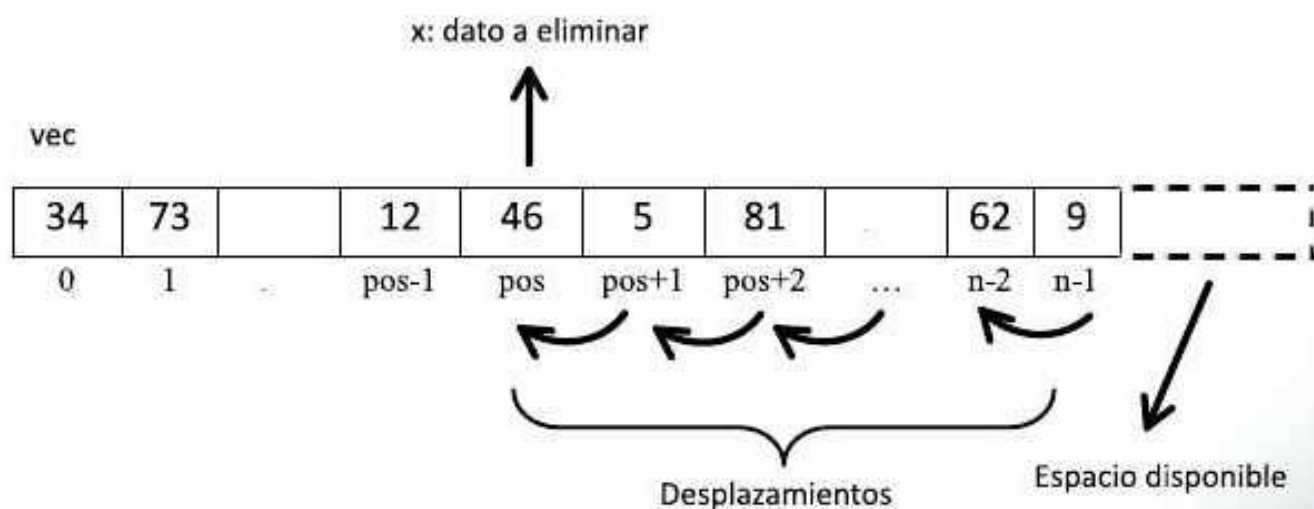


Figura 4.2. Eliminación de un dato en un vector

- El ordenamiento ascendente del vector se lleva a cabo a través del método de la burbuja, que compara el contenido de la primera posición con las demás que le siguen, el contenido de la segunda con todas las que le preceden, y así sucesivamente hasta llegar a comparar el contenido de la penúltima posición con el de la última. Cuando la comparación es verdadera, es decir, cuando se cumple $vec[i].comparar(vec[j]) > 0$ (esto significa que el contenido del vector en la posición i es mayor que el contenido del vector en la posición j), se debe realizar el intercambio de contenido de las dos posiciones, lo que se logra con la variable auxiliar *aux*.

El método de la burbuja tiene varias versiones; aquí se ha presentado una de las más conocidas.

Si se analiza la eficiencia del método *ordenar()* por medio del cálculo de su *orden de magnitud*⁸, se llega a la conclusión que este tipo de ordenamiento es aplicable cuando el volumen de datos a tratar es pequeño, de lo contrario se hace ineficiente. Cuando la cantidad de datos es grande se pueden utilizar otros métodos como el ordenamiento rápido (quick sort), por inserción, selección o montículo (heap sort), que tienen un mejor comportamiento en el tiempo de ejecución. En este sentido, el método *ordenar()* se puede sobrecargar siguiendo los lineamientos lógicos de otros métodos de ordenamiento.

PROBLEMA 15: UNIÓN DE DOS VECTORES

Se tienen dos vectores $v1$ y $v2$ de n y m elementos, respectivamente. Se pide: crear otro vector $v3$ con los elementos correspondientes a la unión de $v1$ y $v2$, ordenar el vector $v3$, e imprimir los tres vectores.

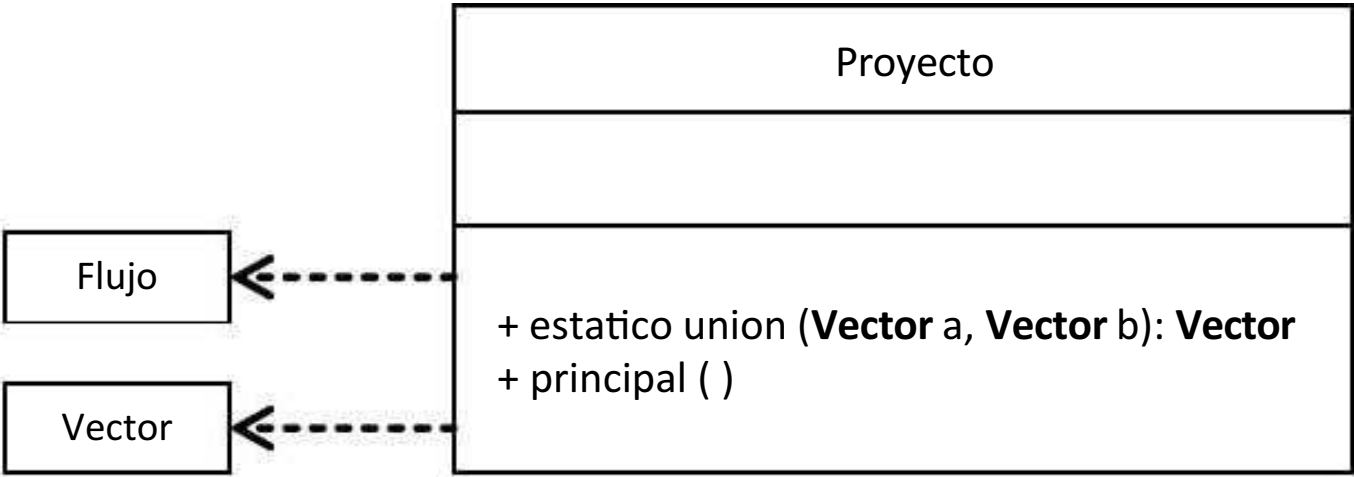
8 El orden de magnitud, conocido también como notación *O* Grande, es una medida para la eficiencia de un algoritmo. El método de la burbuja tiene orden de magnitud cuadrática, expresado como $O(n^2)$. Para mayor información sobre este tema remitirse a [Brassard 1997] y [Aho 1988].

Solución:

a) Tabla de requerimientos

Id. del req.	Descripción	Entradas	Resultados
R1	Conocer el tamaño de los vectores	Dos números enteros m y n	Se conoce el tamaño de los dos arreglos
R2	Conocer los datos del primer vector	m	El vector v1 ha sido creado
R3	Conocer los datos del segundo vector	n	El vector v2 ha sido creado
R4	Unir los vectores v1 y v2	m y n	El vector unión v3 ha sido creado
R5	Ordenar el vector unión	v3.n	El vector v3 está ordenado
R6	Visualizar los tres vectores	v3	Los vectores v1, v2 y v3 han sido impresos

b) Tabla de requerimientos




```

↓
repetir
    m = Flujo.leerEntero( )
hasta (m < 0)
Flujo.imprimir ("Ingrese el valor de n:")
repetir
    n = Flujo.leerEntero( )
hasta (n < 0)

// Reserva de memoria para los dos vectores
Vector v1= nuevo Vector(m)
Vector v2 = nuevo Vector(n)
//Llenar los vectores v1 y v2
v1.llenar( )
v2.llenar( )
// Unir v1 y v2
Vector v3 = nuevo Vector(m + n)
v3 = union(v1, v2)
// Ordenar v3
v3.ordenar( )
// Visualizar el contenido de los tres vectores
v1.mostrar( )
v2.mostrar( )
v3.mostrar( )
fin_metodo
fin_clase

```

PROBLEMA 16: BÚSQUEDA BINARIA RECURSIVA

Escribir un método que busque un elemento dado como parámetro, dentro de un vector que se encuentra ordenado.

Solución:

Pre análisis

La *búsqueda binaria* se aplica a conjuntos de datos ordenados, almacenados en estructuras de datos lineales como vectores o listas enlazadas. Si los datos no se encuentran ordenados, se aplican otros métodos de búsqueda como la secuencial o la hash (por transformación de claves).

En este caso, nos ocuparemos de la búsqueda binaria en un vector, método que formaría parte de la clase *VectorOrdenado*. Un objeto perteneciente a esta clase es un arreglo unidimensional compuesto por elementos del mismo tipo, los cuales se hallan ordenados de forma ascendente (por omisión siempre se asume este tipo de ordenamiento).

Todos los métodos aplicados a la clase *VectorOrdenado* deben conservar el estado de ordenamiento de los datos. Una posible definición para esta clase sería la siguiente.

VectorOrdenado
- vec[]: Objeto + tam: entero + n: entero
+ VectorOrdenado(entero x) + llenarVector(entero x) + mostrarVector() + busquedaBinaria(Objeto x, entero b, entero a): entero + busquedaBinaria(Objeto x): entero + eliminarDato(Objeto x): logico + insertarDato(Objeto x): logico

Los métodos *llenarVector()*, *mostrarVector()*, *eliminarDato()* e *insertarDato()* quedan propuestos como ejercicios para el lector, y no necesariamente se deben tratar como métodos recursivos. Nos interesa por el momento el desarrollo del método recursivo *busquedaBinaria(Objeto x, entero b, entero a):entero*, y el método sobrecargado *busquedaBinaria(Objeto x):entero*, que es de naturaleza no recursiva.

Documentación del método *busquedaBinaria()* recursiva.

Según el diagrama de clases, el método forma parte de la clase *VectorOrdenado*, que a su vez contiene los miembros privados *vec*, *tam* y *n*, donde:


- vec* : nombre del vector ordenado.
- tam*: tamaño del vector, equivalente a la máxima cantidad de elementos que puede almacenar.
- n*: cantidad de elementos reales en el vector. Siempre se cumplirá que $n \leq tam$.

El método *busquedaBinaria(Objeto x, entero b, entero a):entero* retorna la posición *p* donde el dato *x* fue encontrado o retorna un valor igual a -1 si *x* no existe. $0 \leq p < n$

Los parámetros x , b y a adquieren el siguiente significado:

- x : dato a buscar en el vector ordenado.
- b : valor más bajo para un índice del arreglo. Su valor inicial es 0 (cero).
- a : valor más alto para un índice del arreglo. Su valor inicial es el atributo n .

Definición del método:



```

entero busquedaBinaria(Objeto x, entero b, entero a)
    entero p, m

    si (b > a)
        p = -1
    sino
        m = (a + b) / 2
        si (x < vec[m])
            p = busquedaBinaria(x, b, m - 1)
        sino
            si (x > vec[m])
                p = busquedaBinaria(x, m + 1, a)
            sino
                p = m
        fin_si
    fin_si
    retornar(p)
fin_metodo
  
```

Observaciones:

- Los tres parámetros del método son pasados por valor. La primera vez que se ejecute el método los valores de b y a serán 0 y $n-1$, respectivamente. (Recordar que el valor de n es igual a la cantidad de elementos presentes en el vector). El valor del parámetro x siempre será el mismo para las distintas llamadas recursivas.
- El método *busquedaBinaria()* recursiva busca el dato x dividiendo sucesivamente por mitades el vector *vec*. Si el dato x es menor que el dato ubicado en la posición media del vector (dada por m), se busca por la mitad izquierda del

d) Seudo código

```

clase Elecciones
  privado:
    Matriz votos
    Vector vxc
  publico:
    Elecciones( )
    entero f, c
    Flujo.imprimir("Elecciones para alcalde")
    para (f = 0, f < 5, f = f + 1)
      Flujo.imprimir("Zona N° " + (f + 1))
      para (c = 0, c < 4, c = c + 1)
        segun (c)
          caso 0:
            Flujo.imprimir("Candidato A")
            saltar
          caso 1:
            Flujo.imprimir("Candidato B")
            saltar
          caso 2:
            Flujo.imprimir("Candidato C")
            saltar
          caso 3:
            Flujo.imprimir("Candidato D")
        fin_según
      votos.asignarDato(Flujo.leerEntero( ), f, c)
    fin_para
  fin_para
fin_metodo
//-----
resultadoComicios( )
1. entero f, c
2. para (c = 0, c < 4, c = c + 1)
3.   vxc.asignarDato(0, c)
4.   para (f = 0, f < 5, f = f + 1)
5.     vxc.asignarDato(vxc.obtenerDato(c ) +
                       votos.obtenerDato(f, c), c)
6.   fin_para
7. fin_para
8. entero totalV = vxc.obtenerDato(0) + vxc.obtenerDato(1) +
                  vxc.obtenerDato(2) + vxc.obtenerDato(3)
9. real pa, pb, pc, pd //Porcentajes de votos por candidato

```





```

10. pa = (vcx.obtenerDato(0) / totalV) * 100
11. pb = (vcx.obtenerDato(1) / totalV) * 100
12. pc = (vcx.obtenerDato(2) / totalV) * 100
13. pd = (vcx.obtenerDato(3) / totalV) * 100
14. Flujo.imprimir("Total votos candidato A: " +
    vxc.obtenerDato(0) + ", con un porcentaje de: " + pa + "%")
15. Flujo.imprimir("Total votos candidato B: " +
    vxc.obtenerDato(1) + ", con un porcentaje de: " + pb + "%")
16. Flujo.imprimir("Total votos candidato C: " +
    vxc.obtenerDato(2) + ", con un porcentaje de: " + pc + "%")
17. Flujo.imprimir("Total votos candidato D: " +
    vxc.obtenerDato(3) + ", con un porcentaje de: " + pd + "%")
18. si (pa>50) // vxc.obtenerDato(0) > totalV / 2
    Flujo.imprimir("Candidato ganador. " + A)
    sino
19.     si (pb>50)
        Flujo.imprimir("Candidato ganador. " + B)
        sino
20.     si (pc>50)
        Flujo.imprimir("Candidato ganador. " + C)
        sino
21.     si (pd>50)
        Flujo.imprimir("Candidato ganador. " + A)
        sino
            Flujo.imprimir("No hay ganador")
22.         fin_si
23.     fin_si
24. fin_si
25. fin_si
    fin_metodo
//-----
fin_clase // Elecciones

//*****
clase Proyecto
    publico principal( )
        Elecciones e = nuevo Elecciones( )
        e.resultadoComicios( )
    fin_metodo
fin_clase

```

Observaciones:

- El constructor sin argumentos *Elecciones()* es el encargado de llenar la matriz *votos*, miembro privado de la clase *Elecciones*. Este constructor recorre la matriz de votos por filas mediante dos ciclos para unirlos. El ciclo externo imprime un mensaje relativo a la zona; el ciclo interno utiliza un selector múltiple para informar acerca del candidato en proceso.

La instrucción *votos.asignarDato(Flujo.leerEntero(), f, c)* ofrece un particular interés en tanto asigna al miembro privado *mat* de la matriz *votos* en su posición $[f, c]$, y fija el dato digitado por el usuario, obtenido a través de la ejecución de la sentencia *Flujo.leerEntero()*.

- El cuerpo del método *resultadoComicios()* se encuentra numerado para facilitar las siguientes explicaciones:
 - La instrucción 3 asigna 0 (cero) a la posición *c* del vector *vxc*; la instrucción 5, que consume dos líneas de pseudo código, asigna al vector *vxc* en la posición *c*, el contenido de su propia posición más el contenido de la matriz *votos* en la posición $[f, c]$; la instrucción 8 asigna a la variable *totalV* la suma de los votos obtenidos por todos los candidatos.
 - Las instrucciones 10 a 13 asignan a las variables *pa*, *pb*, *pc* y *pd* los porcentajes de votación obtenidos por los candidatos A, B, C y D, respectivamente; las instrucciones 14 a 17 imprimen la cantidad de votos por candidato y su respectivo porcentaje, mientras que las instrucciones 18 a 25 contienen una serie de sentencias si anidadas, que permiten conocer quién fue el candidato ganador de las elecciones o si no hubo ganador.

4.4. EJERCICIOS PROPUESTOS

1. En el problema 16 se presentó la *búsqueda binaria* recursiva sobre un vector ordenado. Sobrecargar el método de la búsqueda binaria de tal manera que se elimine la recursividad (búsqueda binaria no recursiva) y desarrollar el pseudocódigo de los demás métodos de la clase VectorOrdenado, definida así:

VectorOrdenado
- vec[]: Objeto + tam: entero + n: entero
+ VectorOrdenado(entero x) + llenarVector(entero x) + mostrarVector() + busquedaBinaria(Objeto x, entero b, entero a): entero + busquedaBinaria(Objeto x): entero + eliminarDato(Objeto x): logico + insertarDato(Objeto x): logico

El constructor asigna *x* al tamaño del vector, pone *n* en cero e inicializa el vector *vec* con nulos. El método *llenarVector(entero x)* asigna el valor de *x* al atributo *n* y llena el vector con *x* elementos. Los métodos *eliminarDato()* e *insertar dato()* retornan cierto o falso, dependiendo del éxito o fracaso de la operación. Recordar que todas las operaciones deben conservar el ordenamiento del vector.

2. Escriba un método que acepte como parámetros dos vectores ordenados y que retorne un tercer vector ordenado que contenga los elementos de los vectores iniciales.
3. Se tiene una matriz cuadrada de orden *n*. Formar un vector con los elementos de la matriz triangular inferior, ordenar el vector creado e imprimirlo.
4. Redefina la clase *Vector* considerando que el vector puede contener elementos repetidos. Realice los supuestos semánticos necesarios. Por ejemplo, para eliminar

un dato del vector debe aclarar si quita la primera ocurrencia, la última o todas las ocurrencias que se presentan.

5. En un arreglo de enteros, presente algoritmos recursivos para calcular:
 - a) El elemento máximo del arreglo.
 - b) El elemento mínimo del arreglo.
 - c) La suma de los elementos del arreglo.
 - d) El producto de los elementos del arreglo.
 - e) El promedio de los elementos del arreglo.
6. Llenar dos conjuntos. Hallar los conjuntos unión, intersección, diferencia y producto cartesiano entre el primero y el segundo, respectivamente. Recuerde que un conjunto no admite elementos repetidos.
7. Una compañía distribuye N productos distintos. Para ello almacena en un arreglo toda la información relacionada con su mercancía: clave, descripción, existencia, mínimo a mantener de existencia y precio unitario.

Escriba métodos que permitan suplir los siguientes requerimientos:

- a). Venta de un producto: se deben actualizar los campos que correspondan, y comprobar que la nueva existencia no esté por debajo del mínimo (Datos: clave, cantidad vendida).
 - b). Reabastecimiento de un producto: se deben actualizar los datos que correspondan (Datos: clave, cantidad comprada).
 - c). Actualizar el precio de un producto aclarando si el precio aumenta o disminuye (Datos: clave, porcentaje).
 - d). Informar sobre un producto: se deben proporcionar todos los datos relacionados con un producto (Dato. Clave).
 - e). Listado de productos ordenado por precio unitario. El listado incluye clave y precio.
8. El dueño de una cadena de tiendas de artículos deportivos desea controlar sus ventas por medio de una computadora. Los datos de entrada son :

- El número de la tienda (1 a 10)
- Un numero que indica el deporte relacionado con el artículo (1 a 5)
- Costo del artículo.

Al final del día, visualizar el siguiente informe:

- Las ventas totales en el día para cada tienda
- Las ventas totales de artículos relacionados con cada uno de los deportes.
- Las ventas totales de todas las tiendas.

9. El departamento de policía de una ciudad ha acumulado información referente a las infracciones de los límites de velocidad durante un determinado periodo de tiempo. El departamento ha dividido la ciudad en cuatro cuadrantes y desea realizar una estadística de las infracciones a los límites de velocidad en cada uno de ellos. Para cada infracción se ha preparado una tarjeta que contiene la siguiente información:

- Número de registro del vehículo.
- Cuadrante en el que se produjo la infracción.
- Límite de velocidad en milla por hora (mph).
- Velocidad registrada.

a) Elabore métodos para obtener dos informes; el primero debe contener una lista con las multas de velocidad recolectadas, donde la multa se calcula como la suma del costo de la corte (\$20,000) mas \$ 1,250 por cada mph que exceda la velocidad límite. Prepare una tabla con los siguientes resultados:

INFRACCIONES A LOS LÍMITES DE VELOCIDAD

Registro del Vehículo-Velocidad Registrada (MPH)-Velocidad Límite-Multa

b) El segundo informe debe proporcionar un análisis de las infracciones por cuadrante. Para cada uno de los 4 mencionados, debe darse el número de infracciones y la multa promedio.

10. Crear una matriz de orden $m * n$. Imprimir los elementos ubicados en:

- a) La matriz triangular inferior
- b) La matriz triangular superior
- c) Las diagonales principal y secundaria (letra X)

d) El primer y cuarto cuadrante.

1	2
3	4

e) El segundo y tercer cuadrante.

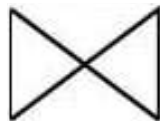
1	2
3	4

f) El reloj de arena



g) El ajedrez (recuerde que no necesariamente es $8 * 8$ sino $m * n$)

h) La mariposa



i) La Letra Z

j) Las montañas



11. Multiplicar dos matrices $A(m, n)$ y $B(n, p)$, ambas de contenido entero. Validar que el número de columnas A sea igual a la cantidad de filas de B , para garantizar el producto.

12. Leer un número en arábigo y mostrarlo en letras.

Ejemplo: 1795 se lee como MIL SETECIENTOS NOVENTA Y CINCO

El programa debe manejar números en un rango de 0 a 99999999

13. Se tienen dos matrices del mismo tipo, $A(m, n)$ y $B(f, c)$. Crear un vector $C(m * n + f * c)$, que contenga los elementos de las matrices A y B recorridas por filas.

4.5 REFERENCIAS

[Aho1998]: Aho-Hopcroft-Ullman. Estructuras de Datos y Algoritmos. Addison-Wesley Iberoamericana, Wilmington, Delaware, E.U.A., 1998.

[Brassard1997]: Brassard-Bratley. Fundamentos de Algoritmia. Prentice Hall, Madrid, 1997.

[Cairó1993]: Cairó-Guardati. Estructuras de Datos. McGraw-Hill, México, 1993.

[Flórez2005]: Flórez Rueda, Roberto. Algoritmos, estructuras de datos y programación orientada a objetos. Ecoe Ediciones, Bogotá, 2005.

[Vásquez2007]: Vázquez, Juan Carlos. Ese inolvidable cafecito. La Coruña, España, <http://www.servicioskoinonia.org/cuentoscortos/articulo.php?num=031>. Consultado en noviembre de 2007.

[Weiss1995]: Weiss, Mark Allen. Estructuras de Datos y Algoritmos. Addison-Wesley Iberoamericana, Wilmington, Delaware, E.U.A., 1995.

CAPÍTULO 5

Relaciones entre clases

- 5.1. Tipos de relación entre clases
 - Asociación
 - Dependencia
 - Generalización / Especialización
 - Agregación y composición
 - Realización
 - Problema 18: Suma de dos números
- 5.2. Paquetes
 - Problema 19: Venta de productos
- 5.3. Ejercicios propuestos
- 5.4 Referencias

Una *clase abstracta* no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es a través de la definición de subclases, que implementan los métodos abstractos declarados. Una clase es *final* si no se pueden definir más subclases a partir de ella. Por ejemplo, la clase *Persona* es una clase abstracta, y las clases *Eprimaria*, *ESecundaria* y *EUniversitario* son finales, según lo muestra la figura 5.2, donde se observa el uso de estereotipos en un diagrama de clases.

Un *estereotipo* es una estructura flexible que se puede utilizar de varios modos. Así, un estereotipo o clisé es una extensión del vocabulario de UML que permite crear nuevos bloques de construcción desde otros ya distintos pero específicos a un problema concreto [Booch1999]; se representa como un nombre entre dos pares de paréntesis angulares. Se puede utilizar el estereotipo sobre el nombre de una clase para indicar algo respecto al papel de la clase [Schmuller2000].

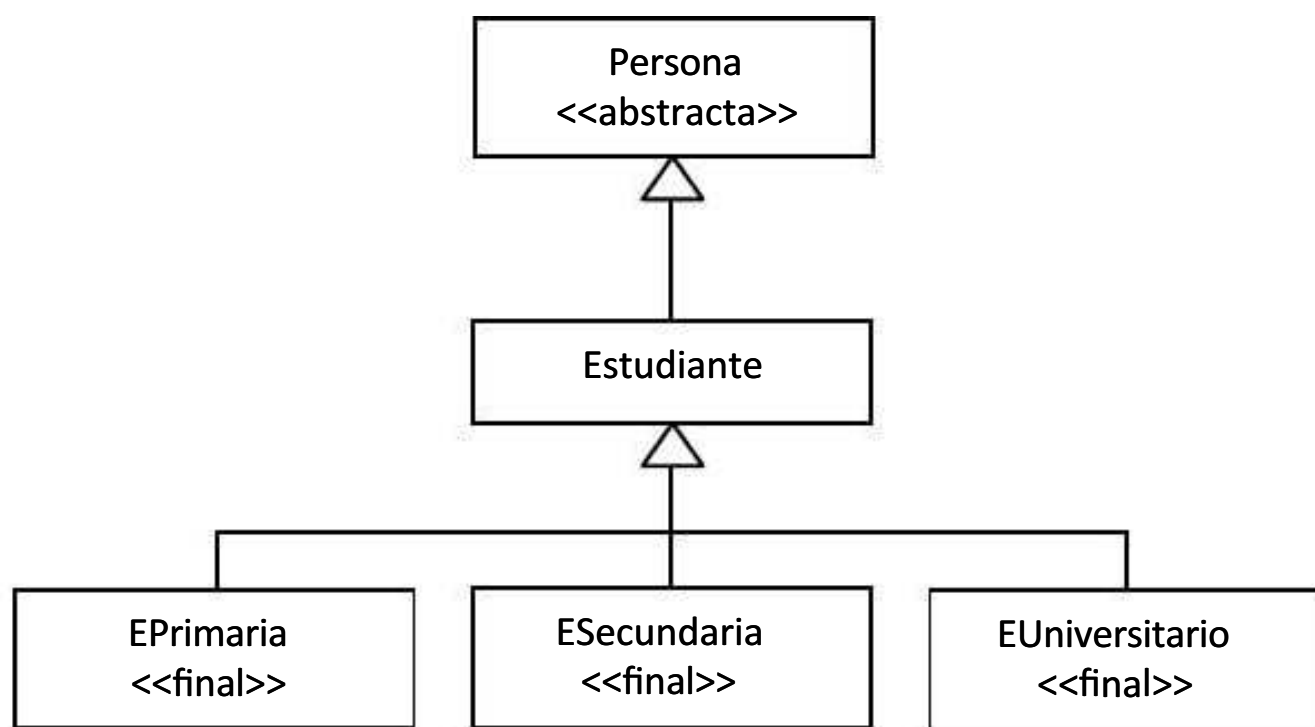


Figura 5.2. estereotipos `<<abstracta>>` y `<<final>>`

En el capítulo 6, los estereotipos y las clases finales se volverán a tratar con el estudio de los mecanismos de herencia.

AGREGACIÓN Y COMPOSICIÓN

Cuando se requiere estructurar objetos que son instancias de clases ya definidas por el analista, se presentan dos posibilidades: *composición* y *agregación*.

La *composición* es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. En este tipo de relación el objeto base se construye a partir del objeto incluido, presentándose así una relación “todo/parte”.

La composición se representa con un rombo relleno que señala hacia el “todo”, como se ilustra en la figura 5.3:

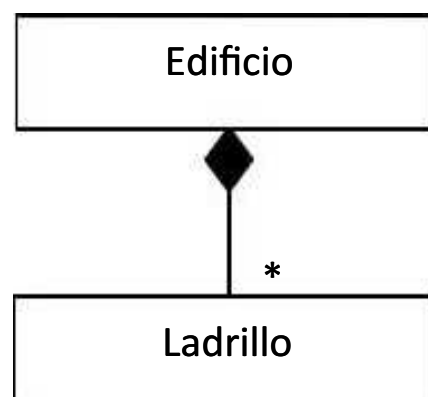


Figura 5.3. Composición

Nota: en la figura 5.3., (*) significa cero o muchos

La *agregación* es un tipo de relación dinámica, donde el tiempo de vida del objeto incluido es independiente del que lo incluye. En este tipo de relación, el objeto base se ayuda del incluido para su funcionamiento. La agregación se representa con un rombo sin relleno que señala hacia el “todo”.

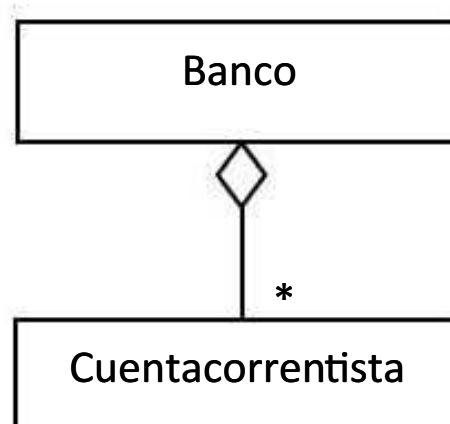


Figura 5.4. Agregación

Se pueden presentar relaciones de *composición* y *agregación* de manera simultánea, como en el siguiente caso:

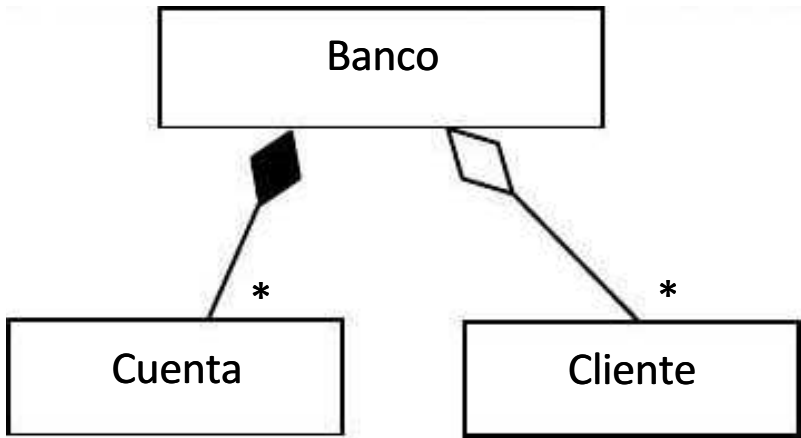
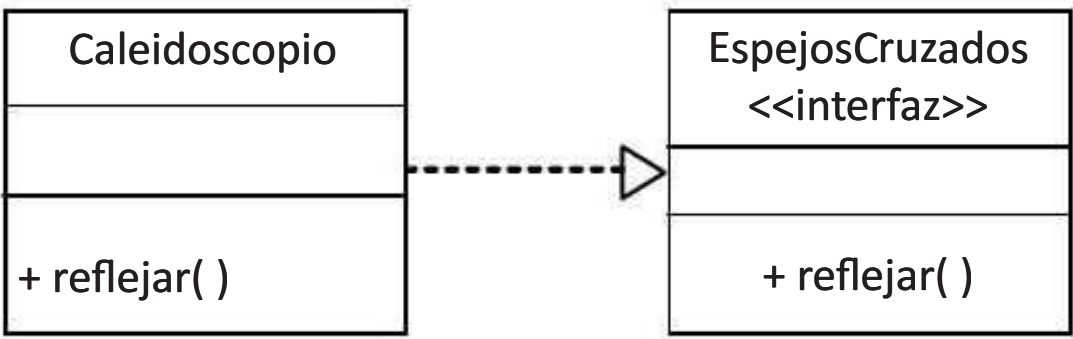


Figura 5.5. Combinación de composición y agregación

REALIZACIÓN

La *realización* es un vínculo semántico entre clases, en donde una clase específica tiene un contrato que otra clase garantiza que cumplirá (clasificación). Se pueden encontrar relaciones de realización entre interfaces y las clases o componentes que las realizan.

Semánticamente, la realización es una mezcla entre dependencia y *generalización*. La realización se representa como la generalización, con la punta de la flecha señalando hacia la interfaz. Por ejemplo, la clase Caleidoscopio realiza interfaz EspejosCruzados.



A continuación, en el problema 18, pueden observarse dos relaciones de dependencia, debido a que el funcionamiento del método principal() depende de las clases Flujo y Suma2.

PROBLEMA 18: SUMA DE DOS NÚMEROS

Sumar dos números enteros. Visualizar el resultado.

Solución:

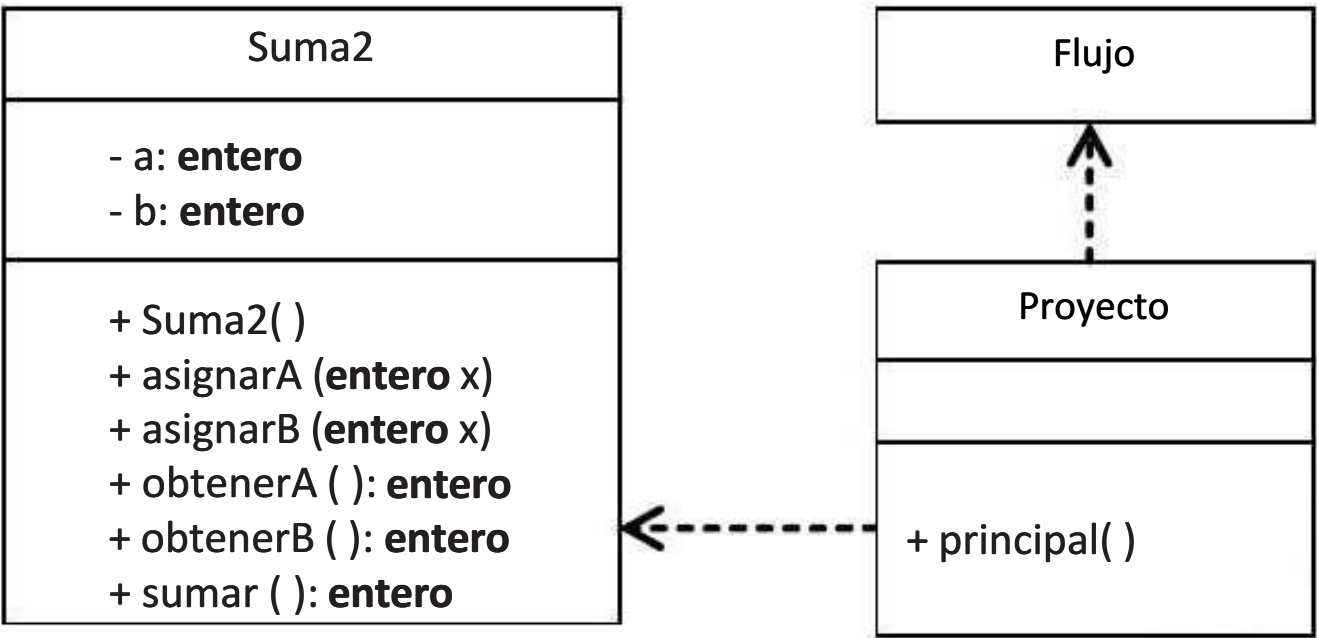
Aunque este problema es de extrema sencillez, puede resolverse dentro del método *principal* () con el uso de dos variables locales, a saber: una operación de suma y la clase Flujo. Aquí se plantea otra alternativa que busca aplicar una relación de uso (dependencia) entre dos clases, veamos:

a) Tabla de requerimientos

La tabla de requerimientos es idéntica a la ya presentada en el Capítulo 1 de este libro.

Identificación del requerimiento	Descripción	Entradas	Resultados (Salidas)
R1	Capturar dos números enteros	Dos números ingresados por el usuario	Dos números enteros almacenados en memoria (variables n1 y n2)
R2	Operar dos números enteros	Las variables n1 y n2	La suma de los números n1 y n2

b) Diagrama de clases



La clase *Suma2* consta de dos atributos (los números a sumar), un constructor, cuatro métodos modificadores y um método analizador.

c) Responsabilidades de las clases
Contratos de la clase *Suma2*

Nombre del método	Requerimientos asociados	Precondición	Postcondición	Modelo verbal
Suma2()	R1	No existen los números a sumar	Los números a sumar existen en un objeto de tipo Sumar2	No aplica
asignarA (entero x)	R1	$a == 0$	El atributo <i>a</i> ha cambiado su estado	Asignar el parámetro <i>x</i> al atributo <i>a</i> (modificar el estado del atributo)
asignarB (entero x)	R1	$b == 0$	El atributo <i>b</i> ha cambiado su estado	Asignar el parámetro <i>x</i> al atributo <i>b</i>
obtenerA(): entero	R2	Se desconoce el valor del atributo <i>a</i> (encapsulado)	Se conoce el valor del atributo <i>a</i>	Recuperar u obtener el estado actual del atributo <i>a</i>
obtenerB(): entero	R2	Se desconoce el valor del atributo <i>b</i> (encapsulado)	Se conoce el valor del atributo <i>b</i>	Recuperar u obtener el estado actual del atributo <i>b</i>
sumar(): entero	R2	No se han sumado los números	Se conoce la suma de los atributos <i>a</i> y <i>b</i>	1. Sumar los valores actuales de los atributos <i>a</i> y <i>b</i> . 2. Retornar la suma

Contrato de la clase Proyecto

Nombre del método	Requerimientos asociados	Precondición	Postcondición	Modelo verbal
principal ()	R1 y R2	Se desconocen los números a sumar	Se ha calculado y visualizado la suma de dos números enteros	1. Instanciar un objeto de la clase Suma2 2. Asignar valores a los atributos a y b del objeto creado 3. Enviar el mensaje sumar() al objeto creado para calcular la suma 4. Visualizar el resultado de la suma

d) Seudo código:

```
clase Suma2
  privado:
    entero a
    entero b
  público:
    //----- métodos para modificar el estado del objeto -----
    asignarA (entero x)
      a = x
    fin_método

    asignarB (entero x)
      b = x
    fin_método

    //----- métodos obtener el estado del objeto -----
    entero obtenerA ( )
      retornar a
    fin_método

    entero obtenerB ( )
      retornar b
    fin_método

    //----- constructores -----
    Suma2 ( )
    fin_método
```



↓

```

Suma2 (entero a1, entero b1)
    asignarA (a1)
    asignarB (b1)
fin_método
//----- función -----
entero sumar ( )
    retornar a + b
fin_método
fin_clase
    
```

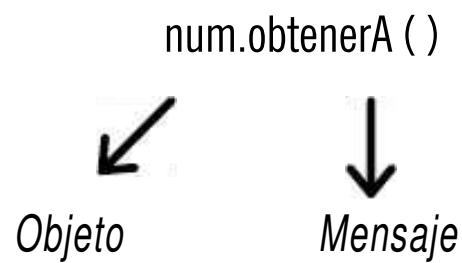
En la clase *Proyecto* se crea un objeto de la clase *Suma2*, tal como se especifica en el contrato de dicha clase:

```

clase Proyecto
    estático principal ( )
        Suma2 num = nuevo Suma2 ( )
        num.asignarA(7)
        num.asignarB(8)
        Flujo.imprimir (num.obtenerA ( ) + " + " + num.obtenerB ( ) +
                        " = " + num.sumar ( ))
    fin_método
fin_clase
    
```

Observaciones:

- La definición dada para la clase *Suma2* (métodos asignar, obtener, constructores y función), le permite al analista-programador una serie de posibilidades para plantear la solución del problema. Ya se ha presentado una primera alternativa: usar el constructor por defecto para crear el objeto *num*, asignarle dos valores a sus miembros dato (7 y 8, valores arbitrarios escogidos a criterio del analista) y mostrar el resultado con tres mensajes al objeto *num*: *obtenerA()*, *obtenerB()* y *sumar()*.



Veamos otras alternativas para sumar dos números enteros, redefiniendo el método principal (), único método que no admite sobrecarga.

Alternativa 1: uso del constructor con argumentos.

```

estático principal ( )
    Suma2 num = nuevo Suma2(15, 24)
    Flujo.imprimir (num.obtenerA ( ) + " + " + num.obtenerB ( ) + " = " +
                    num.sumar ( ))
fin_método
  
```

Alternativa 2: aquí se prescinde del método *sumar*(), es decir, se imprime el resultado de una expresión aritmética.

```

estático principal ( )
    Suma2 num = nuevo Suma2(73, -55)
    Flujo.imprimir (num.obtenerA ( ) + " + ", num.obtenerB ( ) + " = " +
                    num.obtenerA ( ) + num.obtenerA ( ))
fin_método
  
```

Alternativa 3: con entrada de datos especificada por el usuario.

```

estático principal ( )
    Suma2 num = nuevo Suma2 ( )
    Flujo.imprimir ("Ingrese dos números enteros:")
    num.asignarA(Flujo.leerEntero ( ))
    num.asignarB (Flujo.leerEntero ( ))
    Flujo.imprimir (num.obtenerA ( ) + " + " + num.obtenerB ( ) + " = " +
                    num.sumar ( ))
fin_método
  
```


FLORES DE LAS TINIEBLAS

Villers de L'Isle-Adam

A Léon Dierx

"Buenas gentes que pasáis, irogad por los difuntos!"
(Inscripción al borde del camino)

¡Oh, los bellos atardece-
res! Ante los brillantes
cafés de los bulevares, en
las terrazas de las horcha-
terías de moda, ¡qué de mu-
jeres con trajes multi-
colores, qué de elegantes
"callejeras" dándose tono!
Y he aquí las pequeñas
vendedoras de flores,
quienes circulan con sus
frágiles canastillas.

Las bellas desocupadas
aceptan esas flores pe-
recederas, sobrecogidas,
misteriosas...

-¿Misteriosas?

-¡Sí, si las hay!

Existe -sabadlo, sonrien-
tes lectoras-, existe en el
mismo París cierta agencia
que se entiende con varios

conductores en los entierros de lujo, incluso con enterradores, para despojar a los difuntos de la mañana, no dejando que se marchiten inútilmente en las sepulturas todos esos espléndidos ramos de flores, esas coronas, esas rosas que, por centenares, la piedad filial o conyugal coloca diariamente en los catafalcos.

Estas flores casi siempre quedan olvidadas después de las tenebrosas ceremonias. No se piensa más en ello; se tiene prisa por volver. ¡Se concibe!...

Es entonces cuando nuestros amables enterradores se muestran más alegres. ¡No olvidan las flores estos señores! No están en las nubes, son gente práctica. Las quitan a brazadas, en silencio. Arrojarlas apresuradamente por encima del muro, sobre un carretón propicio, es para ellos cosa de un instante.

Dos o tres de los más avispados y espabilados transportan la preciosa carga a unos floristas amigos, quienes, gracias a sus manos de hada, distribuyen de mil maneras, en ramitos de corpiño, de mano, en rosas aisladas inclusive, esos melancólicos despojos.

Llegan luego las pequeñas floristas nocturnas, cada una con su cestita. Pronto circulan incesantemente, a las primeras luces de los reverberos, por los bulevares, por las terrazas brillantes, por los mil y un sitios de placer.

Y jóvenes aburridos y deseosos de hacerse bienquistos por las elegantes, hacia las cuales sienten alguna inclinación, compran esas flores a elevados precios y las ofrecen a sus damas.

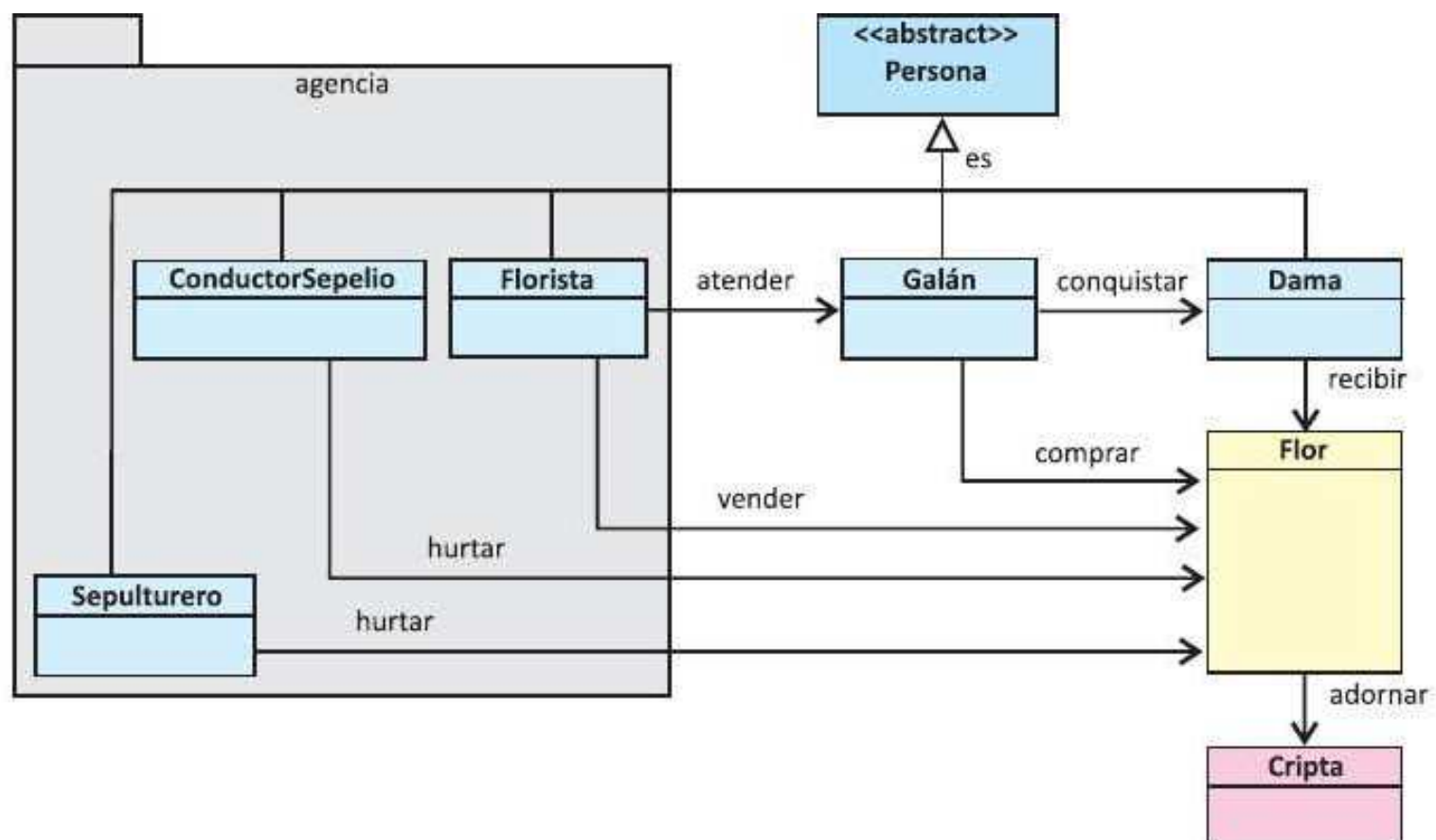
Éstas, todas con rostros empolvados, las aceptan con una sonrisa indiferente y las conservan en la mano, o bien las colocan en sus corpiños.

Y los reflejos del gas empalidecen los rostros.

De suerte que estas criaturas-espectros, adornadas así, con flores de la Muerte, llevan, sin saberlo, el emblema de amor que ellas dieron y el del amor que reciben.



Diagrama de clases:



Objetivos de aprendizaje

- Identificar jerarquías de clase, y dentro de ellas los conceptos de clase base y clase derivada.
- Diferenciar entre clases abstracta, convencional y final.
- Implementar mecanismos de herencia simple y herencia múltiple, esta última mediante interfaces.
- Comprender el concepto de asociación entre clases, roles y cardinalidad (o multiplicidad).
- Aprender a empaquetar clases para casos específicos, en búsqueda de su reutilización para la solución de otros problemas.

6.1. HERENCIA

La *herencia* es una propiedad que permite crear objetos a partir de otros ya existentes. Con ella fácilmente pueden crearse clases derivadas (clase específica) a partir de una *clase base* (clase general); y deja reutilizar a la *clase derivada* los atributos y métodos de la clase base. La herencia conlleva varias ventajas:

- Permite a los programadores ahorrar líneas de código y tiempo en el desarrollo de aplicaciones.
- Los objetos pueden ser contruidos a partir de otros similares, heredando código y datos de la clase base.
- Las clases que heredan propiedades de otra clase pueden servir como clase base de otras, formando así una jerarquía de clases.

La herencia se clasifica en simple y múltiple

HERENCIA SIMPLE

La *herencia simple* le permite a un objeto extender las características de otro objeto y de ningún otro, es decir, solo puede heredar o tomar atributos de un solo padre o de una sola clase.

En el UML, la herencia simple se representa con una flecha de punta triangular. En la figura 6.1., el segmento de línea de la flecha inicia en la clase subordinada y la punta señala hacia la *clase base*.

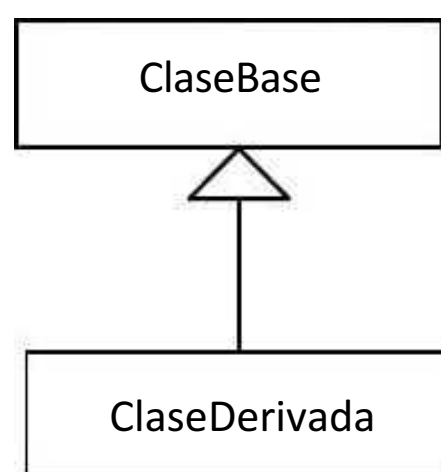


Figura 6.1. Herencia simple

Por ejemplo, en geometría, un rectángulo es un cuadrilátero, así como los cuadrados, los paralelogramos y los trapezoides. Por ende, se puede decir que la clase Rectángulo hereda de la clase Cuadrilátero. En este contexto, el Cuadrilátero es una *clase base* y Rectángulo es una *clase derivada*. [Deitel2007]

A partir de una clase base -o superclase- se pueden definir varias clases derivadas, como se observa en la figura 6.2, equivalente en semántica a la figura 6.3.

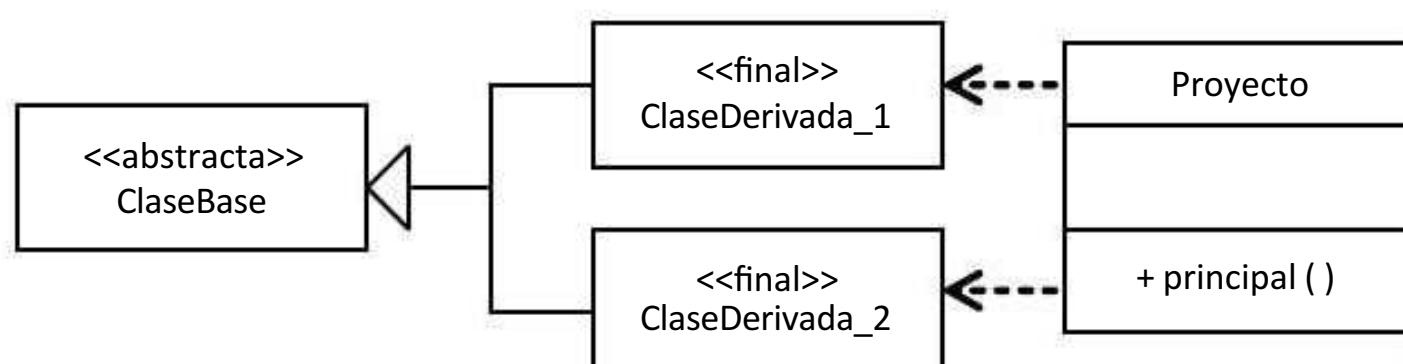


Figura 6.2. Dos clases derivadas de una misma clase base

a aplicar en función del tipo de datos pasado en el parámetro, y es equivalente al concepto de sobrecarga de un método.

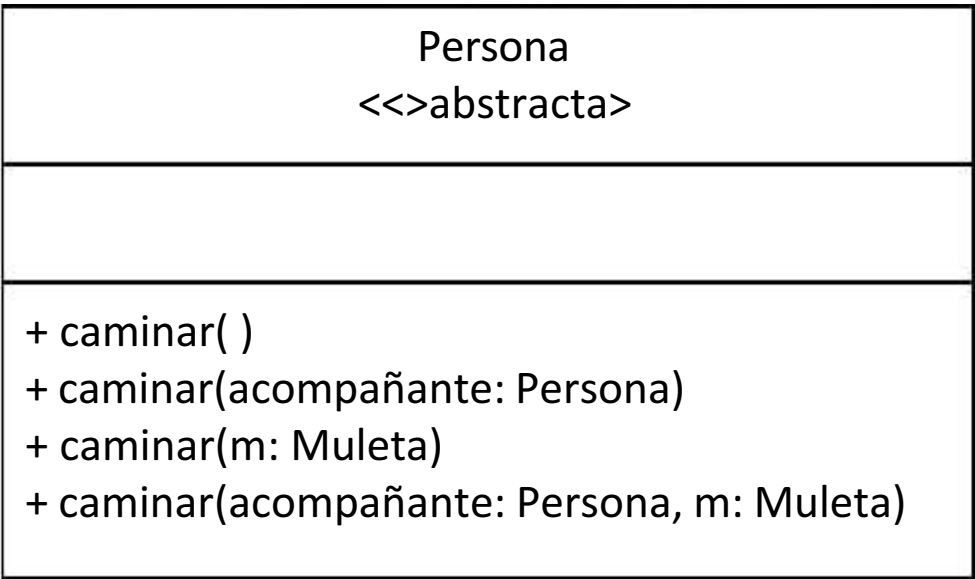


Figura 6.6. Polimorfismo paramétrico

El *polimorfismo de subtipado* permite ignorar detalles de las clases especializadas de una familia de objetos, enmascarándolos con una interfaz común (siendo esta la clase básica). Imagine un juego de ajedrez con los objetos rey, reina, alfil, caballo, torre y peón, cada uno heredando el objeto pieza (ver figura 6.7).

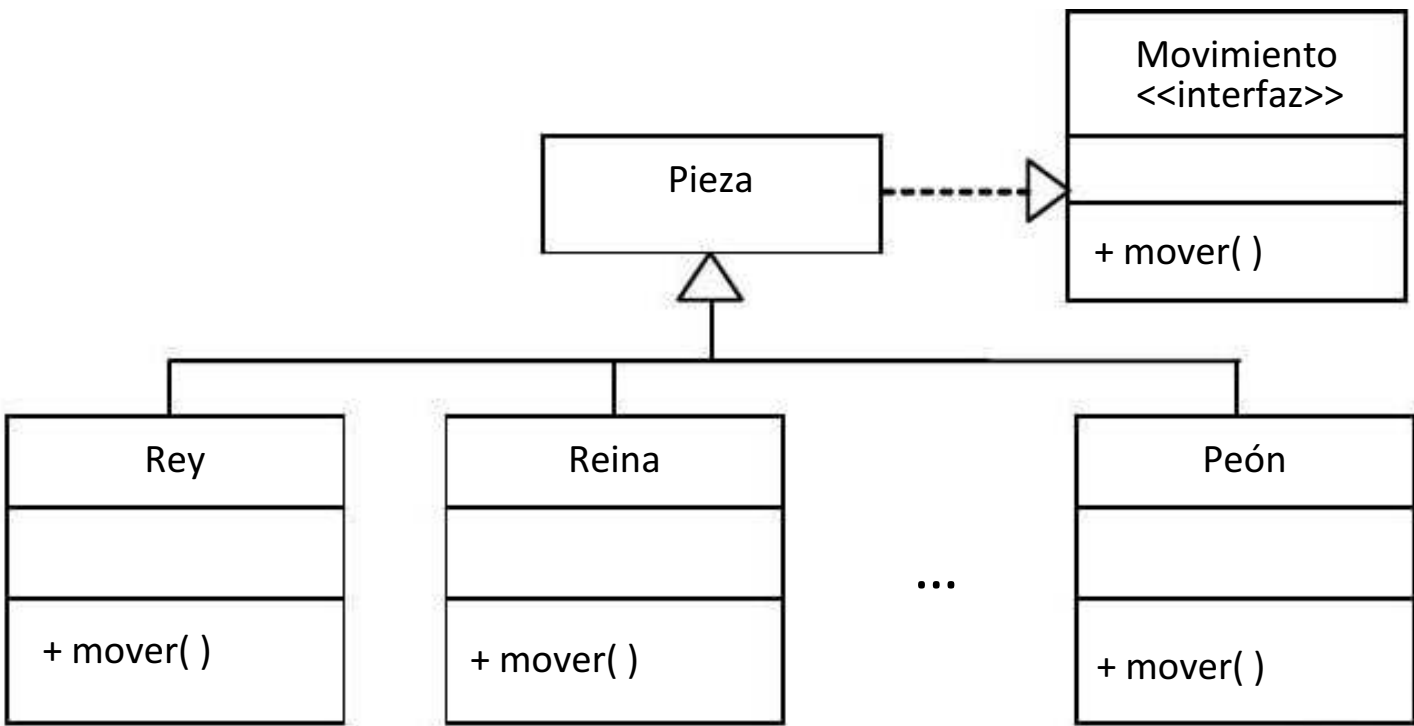


Figura 6.7. Polimorfismo de subtipado

El método *mover()* podría, usando polimorfismo de subtipado, hacer el movimiento correspondiente de acuerdo a la clase de objeto que se llame. Esto permite al programa realizar el movimiento de pieza sin tener que verse conectado con cada tipo de pieza en particular.

6.3. EJERCICIOS PROPUESTOS

1. Con la ayuda de ejemplos de la vida cotidiana, explique los siguientes términos: objeto, clase, herencia simple, herencia múltiple, atributo, método, mensaje, sobrecarga de métodos, polimorfismo y encapsulación de datos.
2. Un centro hospitalario posee N ambulancias distribuidas a lo largo y ancho de una ciudad. De cada ambulancia se conoce su localización. En caso de presentarse una emergencia (por ejemplo un accidente de tránsito o una atención domiciliaria), se toman los datos de su localización.

El problema consiste en hallar la ambulancia más cercana al punto de la emergencia, considerando que ésta puede estar ocupada en otra situación, lo que solicita hallar la siguiente ambulancia más cercana. Se debe enviar un mensaje de SOS al hospital, en caso de que todas las ambulancias estén ocupadas.

3. Defina el paquete tiempo con las clases Fecha y Hora.
La clase Fecha debe incluir los atributos enteros día (entre 1 y 30), mes (entre 1 y 12) y año (mayor o igual a 2006), cuyos valores predeterminados están dados para el 1 de enero de 2006; sobrecargue el constructor con tres parámetros enteros para día, mes y año.

La clase Hora se debe definir con los atributos hor (entre 0 y 23), min (entre 0 y 59) y seg (entre 0 y 59), cuyos valores predeterminados están dados para las 06:30:00; sobrecargue el constructor con tres parámetros enteros para hor, min y seg.

Ambas clases deben constar de las operaciones *asignar* y *obtener* respectivas; las funciones *asignar* deben verificar los rangos de cada atributo.

Incluir además las operaciones:

- `mostrarFecha ()`, que despliega la fecha actual.
- `mostrarHora ()`, que despliega la hora actual.
- `incrementarFecha (entero d, entero m, entero a)`, que incrementa la fecha actual en d días, m meses y a años. Este último método debe finalizar con una fecha consistente.
- `incrementarHora (entero h, entero m, entero s)`, que incrementa la hora actual en h horas, m minutos y s segundos. Este último método debe finalizar con una hora consistente. Sobrecargar este método con un objeto tipo `Hora`, así: `incrementarHora (Hora h)`.

Fuera del paquete tiempo definir la clase `TransacciónBancaria`, con los atributos `numeroCuenta` (tipo cadena), `tipoTransaccion` (tipo entero), `saldo` (tipo real), `fecha` (tipo `Fecha`) y `hora` (tipo `Hora`). Realizar dos transacciones bancarias y visualizar todos sus datos.

Incrementar tanto la fecha de la primera transacción en 3 días, 5 meses y 4 años, como la hora de la segunda transacción con la hora de la primera.

4. El gobierno colombiano desea reforestar los bosques del país según el número de hectáreas que mida cada bosque. Si la superficie del terreno excede a 1 millón de metros cuadrados, entonces decidirá sembrar de la siguiente manera:

Porcentaje de la superficie del bosque	Tipo de árbol
70%	Pino
20%	Oyamel
10%	Cedro

Si la superficie del terreno es menor o igual a un millón de metros cuadrados, entonces la siembra se realizará así:

- Menú Herramientas: Sismoo requiere para su funcionamiento del Kit de Desarrollo de Java (JDK), por lo tanto debe estar instalado previamente en el computador. Esta opción permite configurar el traductor de tal manera que se puedan compilar y ejecutar los programas de la manera correcta.



Figura A.7. Menú Herramientas

Al seleccionarla aparece la siguiente ventana:

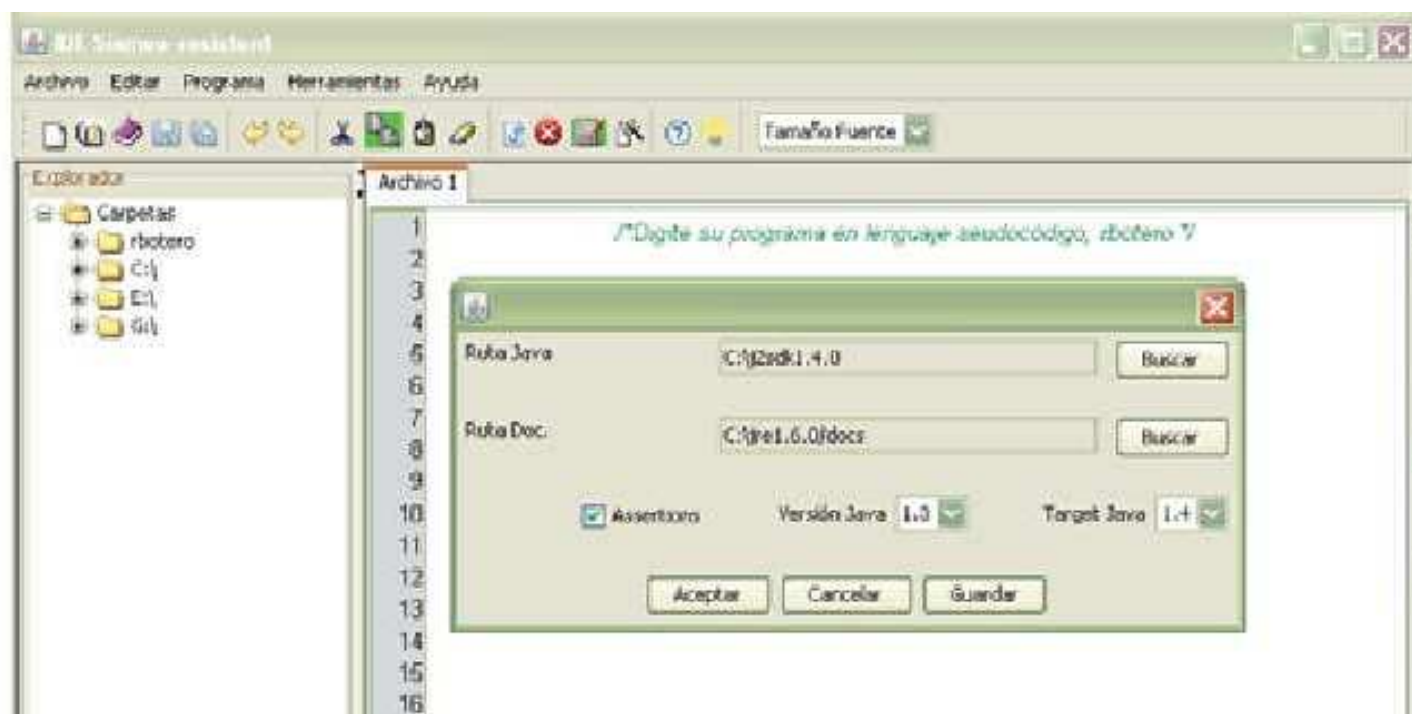


Figura A.8. Configuración de Sismoo

- Menú Ayuda: Permite obtener ayuda en línea acerca del programa SISMOO Resistent e informa acerca de los derechos de autor del programa.



Figura A.9. Menú Ayuda

Aspectos de diseño

Algunos aspectos relacionados con el diseño y construcción de IDE traductor Sismoo se presentan a continuación, en forma grafica, para simplificar su descripción.

En la representación simbólica de SISMOO de la figura A.10, se ilustran los lenguajes que intervienen en el proceso.



Figura A.10. Diagrama simbólica de Sismo

Del diagrama esquemático presentado en la figura A.11, se deduce el proceso de ejecución de un programa desarrollado enseudolenguaje y la relación de Sismo con el lenguaje de programación Java; se observan las labores de compilación y traducción a bytecode (que la realiza la aplicación Javac) y el trabajo de la máquina virtual de Java (JVM) en el manejo de excepciones y generación del programa objeto.

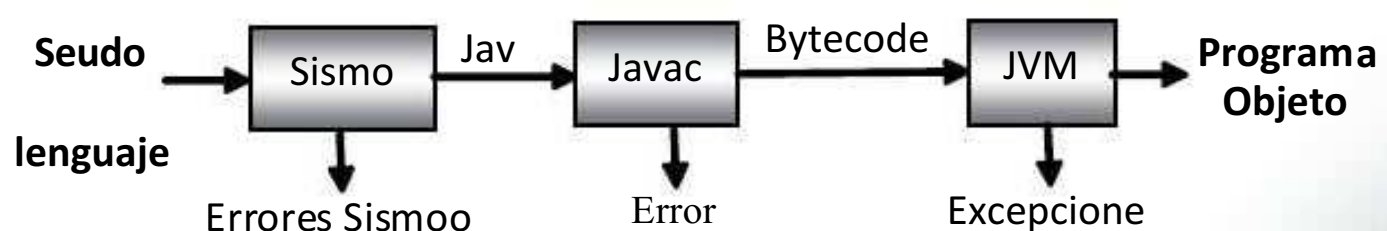


Figura A.11. Diagrama esquemático de Sismo

Para finalizar, en la figura A.12 se ilustran los diagramas de clases conceptuales en notación UML y en la tabla A.1 una breve descripción de cada una de las clases que hacen parte de aplicativo Sismoo.

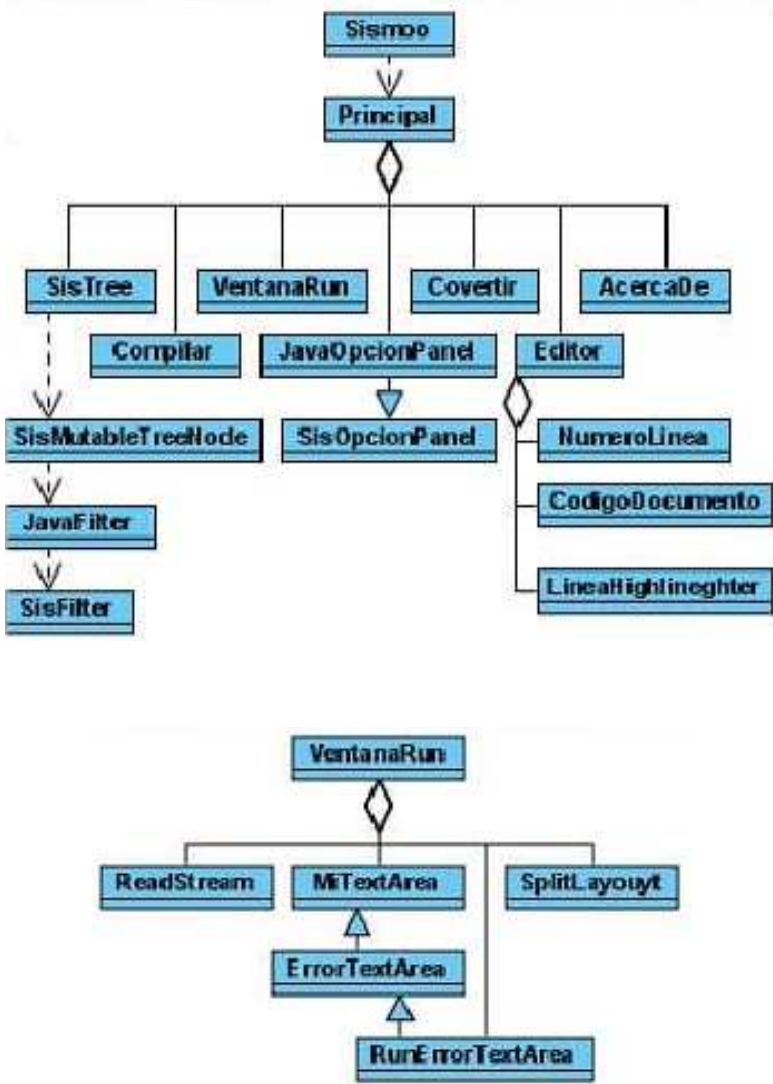


Figura A.12. Diagrama conceptual de clases Sismoo

Tabla A.1 Descripción de clases para Sismoo

Nombre	Deriva de	Breve Descripción
SISMOO	JFrame	Ventana de inicio
Principal	JFrame	Es la clase central que contiene el ambiente de trabajo de SISMOO
SisTree	JTree	Implementa el explorador
SisMutable TreeNode	DefaultMutable TreeNode	Para manejar los nodos o carpetas en el explorador
JavaFilter	FileFilter	Filtro general para los archivos

F

Función, 95

Firma, 38 (V. *también* Prototipo)

G

Generalización, 209, 267

H

Herencia, 209, 212, 237, 264

 simple, 236

 múltiple, 238 (V. *también* Interfaz)

I

Identificación de requerimientos, 128

Identificador, 49

Índice, 168

Interfaz, 238

Interruptor , 142 (V. *también* Bandera)

Iteración, 133 (V. *también* Estructura de control iteración)

M

Matriz, 182

Mensaje, 267

 paso de un, 42

Método, 17, 94

 analizador, 23, 34

 asignarX(), 35

 constructor, 34, 41

 cuerpo de un, 96

 declaración de un, 94

 definición de un, 95

 estático, 20, 71, 98

- finalización, de un 95, 97
- invocación de un, 97
- modificador, 34
- nombre del, 95
- obtenerX(), 35
- principal, 24, 38, 94, 125
- recursivo, 145

Modelo verbal, 17, 115

Multiplicidad, 207 (V. *también* Asociación)

O

Objeto, 31, 94

- estado de un, 32

Operaciones, 17, 91

Operador, 52

- de concatenación, 125
- de resolución de ámbito, 225
- nuevo, 42
- prioridad de un, 54

Operando, 52

Orden de magnitud, 176

P

Palabras clave, 19, 103

Paquete, 23 (V. *también* Espacio de nombres)

- contenedor, 65
- sistema, 20, 65, 74

Paquetes de uso común, 65

Parámetro, 94, 95, 105

- actual, 145
- de entrada, 105
- de entrada y salida, 105
- de salida, 105
- formal, 145, 149

Paso de parámetros, 105

- por dirección, 106
 - por referencia, 106
 - por valor, 106
- Polimorfismo, 246
 - de subtipado, 247
 - paramétrico, 247
- Postcondición, 17
- Precondición, 17
- Procedimiento, 94
- Prototipo, 38
 - definición de, 94 (*V. también Firma*)

R

- Realización, 212
- Recursión
 - directa, 145
 - indirecta, 145
- Relación
 - de agregación, 205 (*V. también Agregación*)
 - de composición, 211
 - de dependencia, 209 (*V. también Dependencia*)
 - de uso, 125
- Responsabilidades de las clases, 17

S

- Selector múltiple, 118 (*V. también Estructura de control selector múltiple*)
- Sentencia
 - retornar, 96
 - según, 120
 - si, 118
- Seudo código orientado a objetos, 18
- Sobrecarga
 - de operadores, 21
 - de métodos, 106

Subclase, 209 (V. *también* Clase derivada)

Subprograma, 94

Superclase, 209 (V. *también* Clase base)

Switch, 142 (V. *también* Bandera)

T

Tabla de requerimientos, 15

Tipo abstracto de datos, 93

Tipo de datos, 47, 269

 cadena, 70

 carácter, 63

 entero, 59

 lógico, 64

 primitivo, 48

 real, 61

Tipo de retorno, 95

V

Variable, 49

 declaración de, 51

 local, 51

Vector, 168

Visibilidad, 93, 95

 privada, 34

 pública, 34

Este libro se procesó con
Microsoft Office Word 2007.

Los diagramas de clase de los cuentos,
epígrafes de algunos capítulos, se realizaron
con una copia de evaluación
de VP-UML EE versión 6.3.

El software Sismoo fue desarrollado en
la edición Java2SE 1.6
Medellín-Colombia, octubre de 2009.