Lógica de programación orientada a objetos cap 7 (Pg 219-287)

Taller de Programación (Pontificia Universidad Javeriana)

Capítulo 7

Orientación a objetos[1]

Ana Isabel Oviedo Carrascal
PhD. Docente de la Universidad Pontificia Bolivariana - sede Medellín.

En los anteriores capítulos de este libro se han trabajado diferentes estructuras de control en un método principal o en métodos estáticos que son invocados desde la misma clase. En este capítulo se utilizarán las mismas estructuras pero en métodos de instancia que no son invocados desde clases sino desde objetos, aunque cumplen con las mismas definiciones de métodos presentadas en el capítulo anterior.

7.1 Definición de clase y objeto

Programar bajo el paradigma de programación orientada a objetos (POO) consiste en simular o modelar los objetos del mundo real. En un establo, por ejemplo, se pueden identificar diferentes animales como entes con características y acciones propias. Como ejemplo introductorio se puede tomar un burro.

1 Video 7. Programación Objetual en el complemento SIL de www.ecoeediciones.com

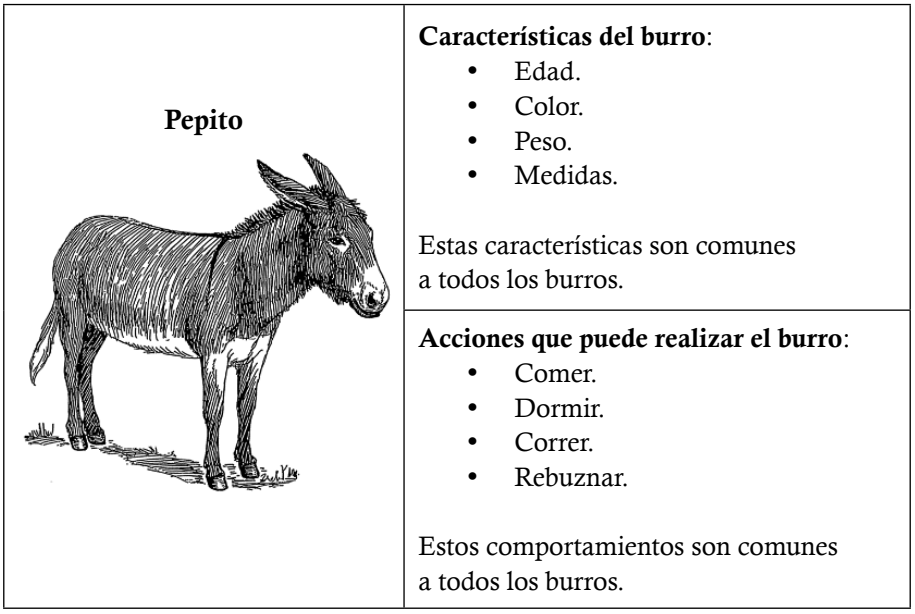


Imagen 7.1. Burro como objeto. Pearson Scott Foresman (Autor). (2007). Donkey (PSF).png [Dibujo digitalizado]. Recuperado el 14 de marzo de 2014, de: http://en.wikipedia.org/wiki/File:Donkey_(PSF).png

El burro en mención (Pepito) es un objeto con características específicas. Claro está, en el establo pueden existir otros burros con diferente edad, color, peso y medidas. De esta manera se puede definir la CLASE Burro como una abstracción a todos los burros existentes en el establo; las características son las VARIABLES MIEMBRO O ATRIBUTOS DE LA CLASE y las acciones son los MÉTODOS MIEMBRO DE LA CLASE o COMPORTAMIENTO DE LA CLASE.

Del anterior ejemplo podemos deducir la definición de clase y objeto. Una *clase* es una plantilla para crear objetos que constituye una abstracción del mundo real, como la clase Burro, la clase persona, la clase cuenta, etcétera. Las clases se nombran como sustantivos en singular y poseen variables que definen la información que se desea almacenar y métodos que definen las acciones que se desean realizar. Las variables se nombran como sustantivos y los métodos como verbos en infinitivo.

Las clases se pueden representar gráficamente mediante un diagrama cuyas especificaciones son establecidas por el lenguaje unificado de modelado *(UML – Unified Modeling Laguage Version 2.4.4)*, el cual permite diseñar visualmente

los sistemas de software. Uno de los diagramas de UML es denominado *diagrama de clases*, donde se utiliza la siguiente representación:

Nombre de la clase
variable1 variable2 …
método1( ) método2( ) …

Así, el diagrama de la clase Burro es:

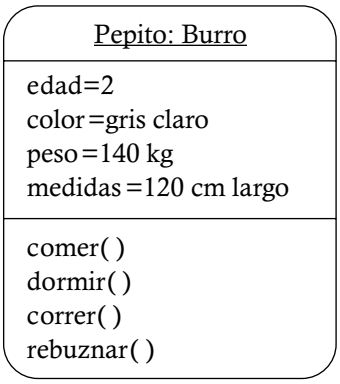
Burro
edad color peso medidas
comer ( ) dormir ( ) correr ( ) rebuznar ( )

Por otro lado, un *objeto* es una instanciación de una clase, es decir, la materialización de la clase. Cuando se instancia un objeto se asignan datos a las variables de la clase y se pueden ejecutar los métodos. Fácilmente se pueden diferenciar las clases de los objetos con el siguiente paralelo: las clases son moldes de galletas y los objetos son las galletas creadas con los moldes. Todas las galletas creadas con el mismo molde son iguales, así que todos los objetos instanciados de la misma clase tienen las mismas variables y los mismos métodos disponibles. Se diferencian en los valores que se asignan a las variables.

Gráficamente, los objetos se representan mediante un *diagrama de objetos* establecido por UML así:

Nombre objeto: Clase
variable1=valor variable2=valor …
método1( ) método2( ) …

El burro Pepito que mencionamos inicialmente es un objeto instanciado de la clase Burro, que podemos representar de la siguiente manera:



7.2 Propiedades de la programación orientada a objetos

La programación objetual define un conjunto de propiedades básicas que los lenguajes de programación orientados a objetos deben cumplir:

- Abstracción.
- Encapsulamiento.
- Ocultamiento de información.
- Sobrecarga.
- Polimorfismo.
- Herencia.
- Reutilización.

En el desarrollo de este capítulo abordaremos detalladamente estas propiedades. Adicionalmente, se introducirán otros conceptos relacionados con el área.

7.3 Abstracción

La abstracción es la característica más básica de la POO y puede definirse como la acción de identificar las cualidades y acciones que un objeto puede realizar, lo que permite diferenciar los conceptos de clase y objeto. Se puede crear la abstracción Burro, compuesta de las características: edad, color, peso y medidas. También se puede crear la abstracción Carro, compuesta de varias características y acciones:

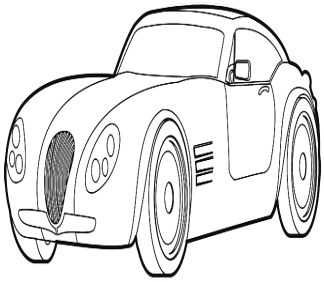
Peugeot 	**Características del carro**: • Marca. • Modelo. • Chasis. • Cantidad de puertas. • Color Estas características son comunes a todos los carros.
	Acciones que puede realizar el carro: • Calcular velocidad máxima. • Calcular tiempo máximo de vida útil. Estos comportamientos son comunes a todos los carros.

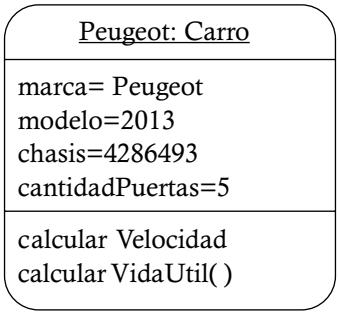
Imagen 7.2. Carro como objeto. OpenClips (Responsable por subir imagen). (2013). Oldtimer Vintage Automobile Vintage Car Classic Car [Imagen digital]. Recuperado el 14 de marzo de 2014, de: http://pixabay.com/en/oldtimer-vintage-automobile-151959

Existen muchos tipos de carros con diferentes marcas, modelos, tipos de chasis y cantidad de puertas diferentes, así que la abstracción Carro agrupa todos los tipos de carros existentes que se pueden definir con esas características. De esta manera tenemos que la abstracción Carro es una clase y todos los tipos de carros que pueden existir con las características definidas en la clase son objetos. El diagrama de la clase Carro es:

Carro
marca modelo chasis cantidadPuertas
calcularVelocidad ( ) calcularVidaUtil ( )

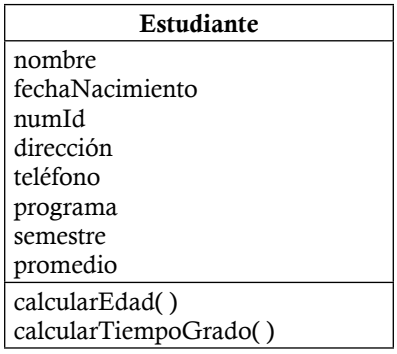
Ejercicio resuelto Cap7-Ejer1
Diagramar el objeto Peugeot como una instancia de la clase Carro presentada anteriormente.

Siguiendo la clase Carro presentada, el objeto Peugeot se puede diagramar de la siguiente manera:

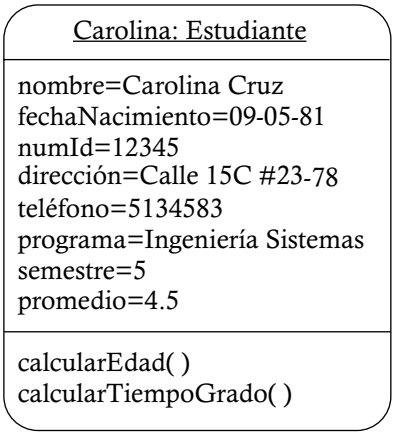


Ejercicio resuelto Cap7-Ejer2
Diagramar una clase que permita representar un estudiante universitario y crear un objeto de dicha clase.

Un estudiante universitario tiene datos básicos que deben ser tenidos en cuenta como: nombre, fecha de nacimiento, número de identificación, dirección, teléfono, programa, semestre actual y promedio crédito. Adicionalmente, se pueden necesitar algunos cálculos básicos de cada estudiante como: calcular edad y calcular tiempo faltante para grado. Con estos datos es posible crear una clase Estudiante así:



De la clase Estudiante es posible instanciar objetos como:



7.4 Encapsulamiento y ocultamiento de la información

El encapsulamiento es la propiedad que tienen las clases de agrupar las características y las acciones relacionadas con una abstracción bajo una misma unidad de programación. Con la encapsulación, las clases son vistas como cajas negras donde se conocen las características y las acciones (variables y métodos) pero no sus detalles internos, es decir que se conoce lo que hace la clase pero no la forma en que lo hace.

Por su parte, el ocultamiento de la información protege los objetos restringiendo y controlando el acceso en la clase que los define. Esto permite al programador definir exactamente cuáles variables y métodos serán visibles para otras clases, asegurándose de que el estado interno de un objeto no pueda ser cambiado de forma inesperada por una entidad externa. Con esta propiedad se logra que los métodos de la clase Burro puedan modificar sin ningún problema las variables de la misma clase, pero no las variables de la clase Carro, a menos que se asigne un permiso específico. De igual manera, los métodos de la clase Carro pueden acceder y modificar las variables de su propia clase, pero no pueden manipular las variables de la clase Burro.

La capacidad de restringir y controlar el acceso en las clases se logra por medio de unos especificadores o modificadores de acceso que se describen a continuación.

7.5 Especificadores o modificadores de acceso

Los especificadores o modificadores condicionan el acceso de las variables, de los métodos de una clase, definiendo su nivel de ocultamiento. Los especificadores pueden ser:

- (+) Público: El elemento puede ser accedido desde cualquier clase. Si es un dato miembro, cualquier clase puede acceder al elemento. Si es un método, cualquier clase puede invocarlo.
- (-) Privado: Solo se puede acceder al elemento desde métodos miembros de la clase donde se encuentra definido, o solo puede invocarse desde otro método de la misma clase.
- (#) Protegido: Proporciona acceso público para las clases derivadas (véase sección de herencia) y acceso privado para el resto de clases.
- ( ) Sin modificador: Se puede acceder al elemento desde cualquier clase que esté en la misma ubicación (carpeta) donde se define la clase.

Las clases no pueden declararse ni protegidas ni privadas, así que sólo pueden declararse púbicas o sin modificador. Por el contrario, las variables y los métodos pueden declararse con cualquiera de los modificadores. Por ejemplo, en la clase Estudiante (Ejercicio resuelto Cap7-Ejer2), se van a incluir los modificadores de acceso de la siguiente manera:

Estudiante
+nombre +fechaNacimiento +numId -dirección -teléfono +programa +semestre -promedio
+calcularEdad( ) -calcularTiempoGrado( )

Como la clase descrita no tiene modificador de acceso, significa que solo puede ser accedida por las clases que se encuentren en la misma ubicación.

Las variables nombre, fechaNacimiento, numId, programa y semestre pueden ser consultadas y modificadas desde cualquier clase, pero las variables dirección, teléfono y promedio solo pueden ser consultadas y modificadas desde métodos de la clase Estudiante. El criterio para definir cuándo una variable

es privada, pública, protegida o sin modificador depende del problema que se va a resolver. Por ejemplo, se podría definir que las variables nombre, fechaNacimiento, numId, programa y semestre no necesitan protección, mientras que las variables dirección, teléfono y promedio tienen información que debería ser accedida con precaución.

Finalmente, el método calcularEdad() puede ser invocado desde cualquier clase. Por el contrario, el método calcularTiempoGrado() solo puede ser invocado desde otro método de la clase Estudiante.

Ejercicio resuelto Cap7-Ejer3
Diagramar una clase que permita representar los datos básicos de una persona y calcular su índice de masa corporal y su edad.

Para cada persona se pueden identificar algunas características básicas como nombre, año de nacimiento, sexo, peso y estatura. Estas constituyen las variables de la clase. Teniendo en cuenta las variables almacenadas para la clase Persona, se pueden crear métodos que permitan calcular el índice de masa corporal y calcular la edad de la persona.

Persona
+nombre +añoNacimiento +sexo -peso -estatura
+calcularIndiceMasaCorporal( ) +calcularEdad( )

La clase Persona puede ser accedida por cualquier clase que se encuentre en la misma ubicación, y las variables nombre, añoNacimiento y sexo pueden ser accedidas desde cualquier clase. Sin embargo, las variables peso y estatura solo pueden ser accedidas desde métodos de la clase Persona. Finalmente, ambos métodos pueden ser accedidos desde cualquier clase.

7.6 Formato de una clase

La conceptualización presentada anteriormente nos permite representar nuestra primera clase. Partiendo del diagrama de clases, el formato tiene la siguiente estructura:

```
MODIFICADOR_ACCESO CLASE Nombre_Clase
    MODIFICADOR_ACCESO TIPO variable1
    MODIFICADOR_ACCESO TIPO variable2
    …
    MODIFICADOR_ACCESO TIPO_RETORNO  metodo1()
                …
    FIN_ metodo1

    MODIFICADOR_ACCESO TIPO_RETORNO  metodo2()
                …
    FIN_ metodo2
FIN_CLASE
```

A continuación se va a crear la clase Persona (Ejercicio resuelto Cap7-Ejer3). El siguiente pseudocódigo codifica la clase Persona:

```
CLASE Persona
    PUBLICO CADENA nombre
    PUBLICO ENTERO añoNacimiento
    PUBLICO CARACTER sexo
    PRIVADO REAL peso
    PRIVADO REAL estatura
    PUBLICO VOID calcularMasaCorporal()
                indice=peso/estatura^2
                ESCRIBA: indice
    FIN_ calcularMasaCorporal
    PUBLICO VOID calcularEdad()
                edad=2013-añoNacimiento
                ESCRIBA: edad
    FIN_calcularEdad
FIN_CLASE
```

En los métodos de la clase Persona se puede observar que no se necesita ingresar como parámetros las variables que son miembro de la clase, ya que todos los métodos de esta pueden acceder directamente a las variables de la clase. Si el método requiere variables adicionales a las que tiene la clase, entonces se pueden ingresar como parámetros o declararlas en su contenido. Sin embargo, la anterior representación de la clase Persona tiene un problema: las

variables de la clase nunca son inicializadas, por lo que contienen valores "basura". Solucionaremos este problema mediante un método llamado constructor.

7.7 Constructores y destructores

El constructor o función constructora es un método público que tiene el mismo nombre de la clase y se ejecuta automáticamente al crear un objeto de la clase. Como característica especial, los métodos constructores no tienen valor de retorno ni parámetros de envío. Este método se emplea para inicializar las variables de la clase o parte de ellas. Los valores utilizados para ello pueden ingresar como parámetros al método constructor o pueden leerse dentro del método. Por ejemplo, para inicializar la clase Persona podemos crear un método constructor que tenga todos los valores para inicializar las variables de la clase como parámetros que ingresan al método:

```
PUBLICO Persona (CADENA n, ENTERO an, CARACTER s, REAL p, REAL e)
    nombre=n
    añoNacimiento=an
    sexo=s
    peso=p
    estatura=e
FIN_persona
```

También podemos crear el método constructor sin parámetros de entrada, y en el contenido del mismo método se leen las variables de la clase que se quieren inicializar:

```
PUBLICO Persona()
  ESCRIBA: "DIGITE EL NOMBRE"
    LEA: nombre
    ESCRIBA: "DIGITE AÑO DE NACIMIENTO"
    LEA: añoNacimiento
    ESCRIBA: "DIGITE EL SEXO"
    LEA: sexo
    ESCRIBA: "DIGITE EL PESO"
    LEA: peso
    ESCRIBA: "DIGITE LA ESTATURA"
    LEA: estatura
FIN_Persona
```

También pueden existir constructores que inicializan solo algunas de las variables, o incluso que no inicializan ninguna de las variables. En estos casos las variables pueden ser inicializadas en otros métodos diferentes al construc-

tor. Ahora, por ejemplo, presentamos un constructor para la clase Persona que inicializa solo algunas variables de la clase:

```
PUBLICO Persona (CADENA n, CARATER s)
    nombre=n
    DIGITE: "AÑO NACIMIENTO"
    LEA: añoNacimiento
    sexo=s
FIN_Persona
```

Este es un constructor que inicializa con valores por defecto:

```
PUBLICO Persona()
   nombre= "    "
   añoNacimiento=0
   sexo= "  "
   estatura=0
FIN_Persona
```

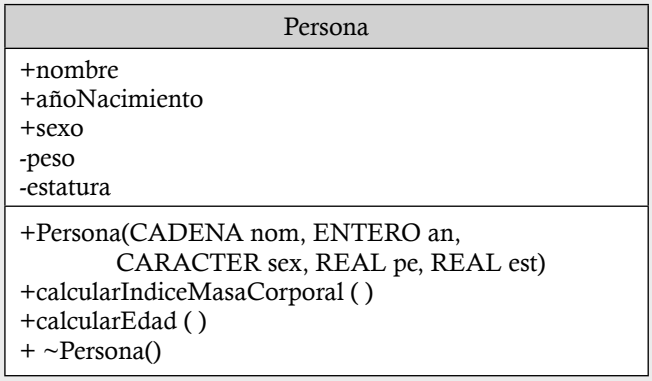
Aquí tenemos un constructor vacío, es decir que no inicializa ninguna de las variables de la clase persona:

```
PUBLICO Persona ()
  ESCRIBA: "CREANDO UN OBJETO DE LA CLASE PERSONA"
FIN_Persona
```

Mientras que el constructor se ejecuta automáticamente al instanciar una clase, existe otro método, llamado destructor, que se ejecuta automáticamente al destruirse la clase. Este método comúnmente es opcional y se utiliza para dar instrucciones finales como liberar memoria, cerrar archivos, desconectarse de bases de datos o simplemente despedirse del usuario. El nombre de este método se forma anteponiendo una virgulilla "~" al nombre de la clase.

```
PUBLICO ~Persona ()
    ESCRIBA: "DESTRUYENDO EL OBJETO"
FIN_~Persona
```

Ejercicio resuelto Cap7-Ejer4
Crear un algoritmo que implemente la clase Persona correspondiente al siguiente diagrama:



Utilizando la codificación de la clase Persona presentada anteriormente, la clase completa con el método constructor y destructor es:

```
CLASE Persona
    PUBLICO CADENA nombre
    PUBLICO ENTERO anoNacimiento
    PUBLICO CARACTER sexo
    PRIVADO REAL peso
    PRIVADO REAL estatura
    PUBLICO Persona (CADENA nom, ENTERO an, CARACTER sex, REAL
    pe, REAL est)
                nombre=nom
                añoNacimiento=an
                sexo=sex
                peso=pe
                estatura=est
    FIN_Persona

    PUBLICO  VOID calcularMasaCorporal()
        REAL indice
                indice =peso/estatura^2
                ESCRIBA: "EL ÍNDICE ES;", indice
    FIN_CalcularMasaCorporal
    PUBLICO VOID calcularEdad()
            ENTERO edad
```

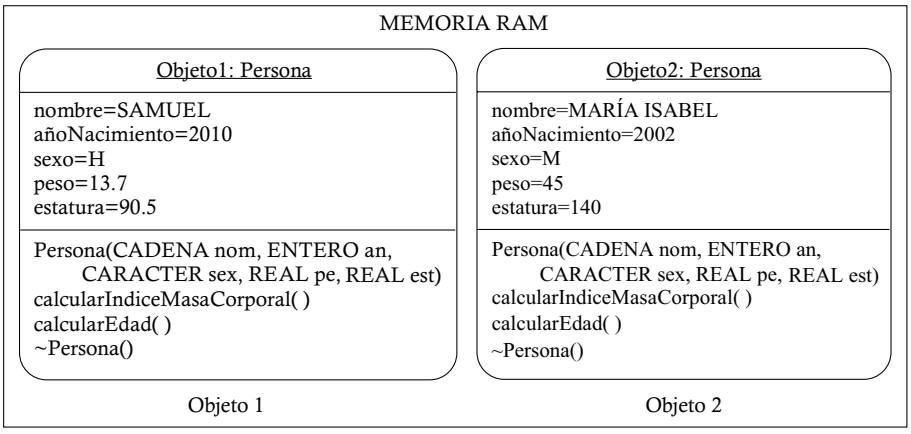
```
            edad=2014-añoNacimiento
            ESCRIBA: "LA EDAD ES:", edad
    FIN_calcularEdad

    PUBLICO ~Persona()
    FIN_~Persona
FIN_CLASE
```

7.8 Creación de objetos

De la clase Persona se pueden crear todos los objetos que sean necesarios, es decir, con la plantilla de la clase se pueden crear muchas personas. Este proceso es llamado instanciación de la clase. Para realizarlo se debe tener en cuenta que el constructor se ejecuta automáticamente; entonces se le deben enviar los argumentos que necesite este método, si los tiene. Para instanciar objetos de la clase Persona tenemos:

```
Persona objeto1, objeto2
objeto1=Persona ("SAMUEL", 2010, "H", 13.7, 90.5)
objeto2=Persona ("MARÍA ISABEL", 2002, "M", 45, 140)
```

En la primera línea se están definiendo las dos variables que almacenarán los objetos, y en las dos siguientes se está invocando al constructor de la clase para que cree los objetos. Al instanciar estos dos objetos, en la memoria RAM se separa espacio para los dos objetos compuestos por todas las variables y los métodos de la clase Persona:



7.9 Definición del método principal

Una vez se han definido las clases necesarias para solucionar un problema, se puede proceder a usarlas en el método principal. Usualmente, es en este método donde se realiza el proceso de instanciación y se procede a utilizar las variables y los métodos de las clases.

Dentro del método principal es posible cambiar o imprimir el valor de las variables de los objetos creados. Esta manipulación se realiza por medio del operador punto (.) siempre que las variables hayan sido declaradas como públicas:

objeto.variable

Por ejemplo, si es necesario cambiar el nombre del objeto1, entonces se realiza la siguiente operación:

objeto1.nombre="DAVID"

Si se requiere mostrar por pantalla el sexo del objeto2, entonces:

ESCRIBA: objeto2.sexo

Por otro lado, para utilizar los métodos de las clases, la invocación se realiza desde el objeto por medio del operador punto (.) de la forma:

- Si el método es una función que retorna valor:
  retorno=objeto.Metodo()
- Si no retorna valor:
  objeto.Metodo()

Por ejemplo, siguiendo con la clase Persona y los objetos instanciados de esta clase, se puede invocar el método calcularEdad() del objeto2 con la siguiente instrucción:

objeto2.CalcularEdad()

Es de anotar que este método no tiene valor de retorno y produce una impresión por pantalla:

12

Finalmente, se obtiene un método principal con las siguientes instrucciones:

```
METODO PRINCIPAL()
    Persona objeto1, objeto2
    objeto1=Persona (“SAMUEL”, 2010, “H”, 13.7, 90.5)
    objeto2=Persona (“MARÍA ISABEL”, 2002, “M”, 45, 140)
    objeto1.nombre=”DAVID”
    ESCRIBA: “EL SEXO ES:”, objeto2.sexo
    objeto2.calcularEdad()
FIN_PRINCIPAL
```

En la primera instrucción del método principal se declaran los dos objetos. En las siguientes dos instrucciones se le envían argumentos a la función constructora para inicializar los dos objetos (darles valores a cada una de sus variables). Estas dos instrucciones se pueden simplificar en una sola así:

```
Persona objeto1=Persona (“SAMUEL”, 2010 , “H”, 13.7, 90.5)
Persona objeto2= Persona (“MARÍA ISABEL”, 2002, “M”, 45, 140)
```

La ejecución de este método principal produce la siguiente impresión por pantalla:

```
EL SEXO ES: M
LA EDAD ES: 12
```

7.10 Pasos para solucionar un problema mediante POO

Para solucionar un problema mediante POO se deben seguir los siguientes pasos:

1. **Identificación de datos, acciones y limitaciones:** Se debe revisar cuidadosamente la descripción del problema para identificar los datos de entrada, los datos de salida, los procesos o acciones requeridas y las restricciones existentes.
2. **Definición de clases:** Para definir las clases que se deben implementar es necesario realizar las siguientes actividades:
   - Identificar las posibles clases buscando los sustantivos relevantes dentro del enunciado que no sean datos de entrada o de salida.
   - Relacionar los sustantivos con los datos de entrada, los datos de salida y las acciones identificadas.

- Seleccionar como clases aquellos sustantivos que tienen asignado al menos un dato o al menos una acción. Los demás sustantivos son eliminados. Los datos asignados se convierten en las variables miembro y las acciones asignadas se convierten en los métodos miembro de las clases.
- Cada clase debe poseer al menos un método constructor. Se sugiere utilizar siempre este método para inicializar todos o algunos de los datos miembro de la clase. El método destructor es opcional.
- Nombrar cada clase con un sustantivo en singular y diagramarla con las variables, los métodos y sus respectivos modificadores de acceso.

3. **Definición del método principal:** Se debe analizar cuántas instancias de las clases son necesarias, es decir, cuántos objetos deben ser creados de cada clase. Igualmente, se debe identificar cuáles métodos deben ser invocados para cada objeto.
4. **Creación de pseudocódigo:** Este es el paso final para solucionar un problema mediante POO. Se debe partir del diagrama de clases.
5. **Prueba de escritorio:** Este paso es muy importante para verificar que los datos de salida sean correctos.

Ejercicio resuelto Cap7-Ejer5
Construir un programa que permita realizar las operaciones de suma, resta, multiplicación y división de dos números ingresados por el usuario.

1. Identificación de datos, acciones y limitaciones
- Datos de entrada: número1 y número2.
- Datos de salida: resultado de las operaciones.
- Acciones: suma, resta, multiplicación y división.
- Limitaciones: Todas las operaciones se realizan con los mismos números.

2. Definición de clases
- Identificación de posibles clases (sustantivos relevantes): operaciones, números, usuario.
- Relación de los sustantivos con los datos y las acciones:
- Operaciones: número 1, número 2, sumar, restar, multiplicación y división.
- Números: N/A.
- Usuario: N/A.

Continúa en la siguiente página

- Clase seleccionada: Operacion.
- Definición de constructores y destructores para cada clase:
- Operacion(): Este constructor lee los dos datos miembro de la clase.
- Diagramación de clases con sus variables y métodos:

Operacion
-numero1 -numero2
+Operacion() +sumar(): REAL +restar(): REAL +multiplicar(): REAL +dividir()

- Explicación de los métodos:
  Operacion: Constructor que lee las variables de la clase.
  sumar: Calcula la suma de las variables de la clase y retorna el resultado de tipo REAL.
  restar: Calcula la resta de las variables y retorna el resultado de tipo REAL.
  multiplicar: Calcula la multiplicación de las variables y retorna el resultado de tipo REAL.
  dividir: Calcula la división de las variables y muestra el resultado por pantalla.

3. Definición del método principal

Se debe crear un objeto de la clase Operacion que permita invocar los diferentes métodos de la clase.

Definición de variables:

objeto: Objeto de la clase Operacion para invocar los métodos de la clase.
opcion: Variable para controlar la selección del menú.
respuesta: Variable para almacenar los resultados de las operaciones.

Algoritmo

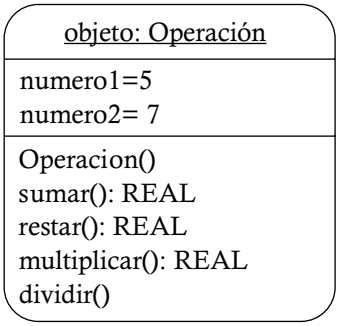
```
CLASE Operacion
    PRIVADO REAL numero1
    PRIVADO REAL numero2
```

```
        PUBLICO Operacion()
          ESCRIBA: "DIGITE EL PRIMER NÚMERO"
          LEA: numero1
          ESCRIBA: "DIGITE EL SEGUNDO NÚMERO"
          LEA: numero2
          FIN_Operacion
          PUBLICO REAL sumar()
                REAL resultado
                 resultado=numero1+numero2
                RETORNE resultado
        FIN_sumar
        PUBLICO REAL restar()
                REAL resultado=numero1 - numero2
                RETORNE resultado
        FIN_restar
PUBLICO REAL multiplicar()
                REAL resultado=numero1*numero2
                RETORNE resultado
        FIN_multiplicar
        PUBLICO VOID dividir()
                SI (numero2==0) ENTONCES
                        ESCRIBA: "ERROR EN LA OPERACIÓN"
                SI_NO
                        REAL resultado=numero1/numero2
                        ESCRIBA: "DIVISIÓN =", resultado
                FIN_SI
        FIN_dividir
METODO PRINCIPAL ()
   ENTERO opcion
  REAL respuesta
       Operación objeto=Operacion()
   HAGA
      ESCRIBA: "MENÚ"
      ESCRIBA: "1: SUMAR"
      ESCRIBA: "2: RESTAR"
      ESCRIBA: "3: MULTIPLICAR"
      ESCRIBA: "4: DIVIDIR"
      ESCRIBA: "5: SALIR DEL MENÚ"
      ESCRIBA: "DIGITE OPCIÓN "
      LEA: opcion
      CASOS DE (opcion)
          CASO 1: respuesta=objeto.sumar()
                ESCRIBA: "LA SUMA ES:", respuesta
          CASO 2: ESCRIBA: "LA RESTA ES:", objeto.restar()
          CASO 3: ESCRIBA: "LA MULTIPLICACION ES:", objeto.multiplicar()
          CASO 4: objeto.dividir()
```

```
    FIN_CASOS
  MIENTRAS (opcion<>5)
 FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Si los valores ingresados en el constructor son 5 y 7, entonces el objeto tendrá los siguientes valores en sus variables de clase:



Salida

```
LA SUMA ES = 12
LA RESTA ES = -2
LA MULTIPLICACIÓN ES= 35
DIVISIÓN =  0,71
```

Ejercicio resuelto Cap7-Ejer6

Construir un programa que permita manipular la cuenta bancaria de una persona. La información que se tiene de la cuenta es el número, la clave de acceso, el dueño y el saldo. Las operaciones que se pueden realizar con la cuenta son: consignar dinero, retirar dinero, cambiar la clave y consultar el saldo. Se debe crear la cuenta de Magdalena Sánchez, con número 1012, clave 1234 y saldo en cero. A la cuenta en mención se le consignan $100 000. También se debe crear la cuenta de Joselito Pérez, con número 1013, clave 9876 y saldo $500 000, quien cambia su clave y consulta su saldo.

Continúa en la siguiente página

1. Identificación de datos, acciones y limitaciones

- Datos de entrada: número, clave, dueño y saldo.
- Datos de salida: saldo y clave.
- Acciones: consignar dinero en la cuenta, retirar dinero de la cuenta, cambiar la clave y consultar el saldo.
- Limitaciones: Una persona solo tiene asignada una cuenta; no se valida la existencia de saldo negativo; no se valida que una clave sea válida.

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): cuenta, persona.
- Relación de los sustantivos con los datos y las acciones:
- Cuenta: número, clave, nombre, saldo, consignar, retirar, cambiar clave y consultar saldo.
- Persona: N/A.
- Clase seleccionada: Cuenta.
- Definición de constructor para la clase:
- Cuenta (ENTERO num, ENTERO cla, CADENA nom, REAL sal): Este constructor recibe como parámetros todos los datos.
- Diagramación de clases con sus variables y métodos:

Cuenta
-numero -clave -nombre -saldo
+Cuenta(ENTERO num, ENTERO cla, CADENA nom, REAL sal) +consultarSaldo() +consignarDinero(REAL valorConsignacion) +retirarDinero(REAL valorRetiro) +cambiarClave(ENTERO nuevaClave)

Explicación de los métodos:

Operacion: Constructor que lee las variables de la clase.

consultarSaldo: Muestra por pantalla el valor de la variable saldo.

consignarDinero: Aumenta el saldo con un valor ingresado al método e invoca el método de consultar saldo.

Continúa en la siguiente página

retirarDinero: Valida si se puede realizar el retiro, disminuye el saldo con un valor ingresado al método y finalmente invoca al método consultarSaldo.
cambiarClave: Cambia el valor de la clave con un entero que ingresa al método.

3. Definición del método principal
Se deben crear dos objetos de la clase Cuenta: uno para Magdalena Sánchez y otro para Joselito Pérez.

Definición de variables:
nuevaClave: Variable para almacenar la nueva clave que va a ser cambiada en el objeto.
obj1: Objeto de tipo Cuenta para almacenar los datos de Magdalena.
obj2: Objeto de tipo Cuenta para almacenar los datos de Joselito.

Algoritmo

```
CLASE Cuenta
    PRIVADO ENTERO numero
    PRIVADO ENTERO clave
    PRIVADO CADENA nombre
    PRIVADO REAL saldo
PUBLICO Cuenta (ENTERO num, ENTERO cla, CADENA  nom, REAL sal)
            NUMERO=num
            CLAVE=cla
            NOMBRE=nom
            SALDO=sal
    FIN_CUENTA
    PUBLICO VOID consultarSaldo()
            ESCRIBA: "SU SALDO ES", saldo
    FIN_consultarSaldo
PUBLICO VOID consignarDinero (REAL valorConsignacion)
            saldo=saldo+valorConsignacion
            ESCRIBA: "CONSIGNACIÓN EXITOSA"
            consultarSaldo()
    FIN_consignarDinero
    PUBLIC VOID retirarDinero (REAL valorRetiro)
            SI (saldo>=valorRetiro) ENTONCES
                    saldo=saldo-valorRetiro
                    ESCRIBA: "RETIRO EXITOSO"
                    consultarSaldo()
             SI_NO
```

```
                    ESCRIBA: "FONDOS INSUFICIENTES"
                FIN_SI
        FIN_retirarDinero
        PUBLICO VOID cambiarClave(ENTERO nuevaClave)
                clave=nuevaClave
                ESCRIBA: "LA CLAVE HA SIDO CAMBIADA CON ÉXITO"
        FIN_cambiarClave
        METODO PRINCIPAL ()
          ENTERO nuevaClave
              Cuenta obj1= Cuenta (1012,1234, "MAGDALENA SÁNCHEZ",0)
              Cuenta obj2= Cuenta (1013,9876, "JOSELITO PÉREZ",500000)
            ESCRIBA: "DIGITE LA NUEVA CLAVE:"
            LEA: nuevaClave
            Cuenta obj1.consignar(100000)
                obj2.cambiarClave(nuevaClave)
                obj2.consultarSaldo()
        FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Si en el método cambiarClave() del objeto obj2 se ingresa la clave 5555, entonces los objetos creados tendrán los siguientes valores después de ejecutarse el método principal:

obj1:Cuenta
numero=1012 clave=1234 nombre=MAGDALENA SÁNCHEZ saldo=100 000
Cuenta(ENTERO num, ENTERO cla, CADENA nom, REAL sal) consultarSaldo() consignar Dinero(REAL valorConsignacion) retirar Dinero(REAL valorRetiro) cambiarClave(ENTERO nuevaClave)

obj2:Cuenta
numero=1013 clave=5555 nombre=JOSELITO P ÉREZ saldo=500 000
Cuenta(ENTERO num, ENTERO cla, CADENA nom, REAL sal) consultarSaldo() consignar Dinero(REAL valorConsignacion) retirar Dinero(REAL valorRetiro) cambiarClave(ENTERO nuevaClave)

Salida

```
CONSIGNACIÓN EXITOSA
SU SALDO ES 100000

LA CLAVE HA SIDO CAMBIADA CON ÉXITO
SU SALDO ES 500000
```

Ejercicio resuelto Cap7-Ejer7

Construir un programa que calcule el promedio crédito de un estudiante. Para cada materia se debe ingresar la nota y los créditos. El programa debe informar el nombre del estudiante, el semestre actual y el promedio crédito.

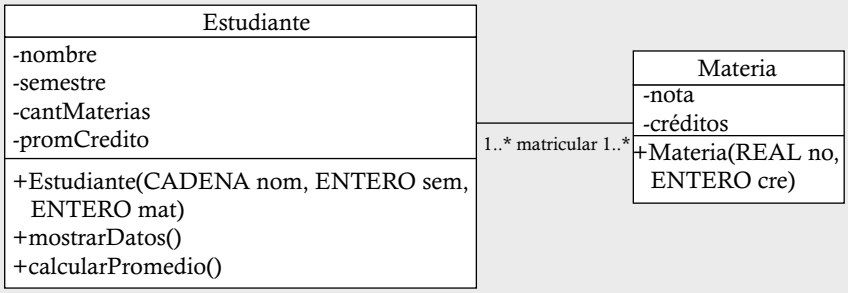
1. Identificación de datos, acciones y limitaciones

- Datos de entrada: nombre estudiante, semestre, cantidad de materias, nota de cada materia y créditos de cada materia.
- Datos de salida: nombre del estudiante, semestre actual y promedio crédito.
- Acciones: calcular el promedio crédito.
- Limitaciones: Solo se hace el cálculo para un estudiante.

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): estudiante, materia.
- Relación de los sustantivos con los datos y las acciones:
- Estudiante: nombre estudiante, semestre, cantidad de materias y promedio crédito.
- Materia: nota de cada materia y créditos de cada materia.
- Clases seleccionadas: Estudiante y Materia.
- Definición de constructores y destructores para cada clase:
- Estudiante (CADENA nom, ENTERO sem, ENTERO mat): Este constructor recibe como parámetros todos los datos, excepto el promedio, que es calculado por un método de la clase.
- Materia (REAL no, ENTERO cre): Este constructor recibe como parámetros todos los datos de la clase Materia.
- Diagramación de clases con sus variables y métodos: La relación entre las clases es representada por medio de una asociación que se lee de la siguiente forma:
  - Un estudiante activo tiene matriculadas una o muchas materias.
  - Una materia activa tiene matriculados uno o muchos estudiantes.

Continúa en la siguiente página



- **Explicación de los métodos de la clase Estudiante:**
  Estudiante: Constructor que recibe las variables de la clase.
  mostrarDatos: Método que muestra por pantalla los datos de la clase Estudiante.
  calcularPromedio: Método que ingresa la información del estudiante, incluyendo los datos de las materias que tiene matriculadas.
- **Explicación de los métodos de la clase Materia:**
  Materia: Constructor de la clase que recibe los datos.

Definición del método principal

Se debe ingresar el nombre del estudiante, el semestre y la cantidad de materias para enviárselos al constructor de la clase Estudiante. Después de instanciar el objeto de la clase Estudiante se debe invocar el método que se encarga de calcular el promedio. No es necesario instanciar objetos de la clase Materia ya que estos objetos se crean dentro del método que calcula el promedio en la clase Estudiante.

Definición de variables:

obj: Objeto de la clase Estudiante desde el cual se invocan los métodos de la clase.
nom: Variable que almacena el nombre del estudiante.
sem: Variable que almacena el semestre actual del estudiante.

Algoritmo

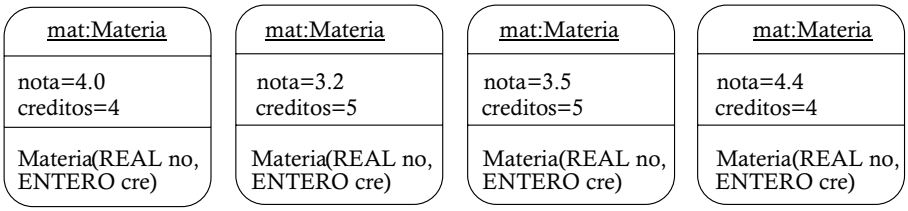
```
CLASE Materia
    PUBLICO REAL nota
    PUBLICO ENTERO creditos
 PUBLICO Materia (REAL no, ENTERO cre)
    nota=no
```

```
        creditos=cre
        FIN_Materia
    FIN_CLASE

    CLASE Estudiante
        PRIVADO CADENA nombre
        PRIVADO ENTERO semestre
        PRIVADO ENTERO cantMaterias
        PRIVADO REAL promCredito

        PUBLICO ESTUDIANTE (CADENA nom, ENTERO sem, ENTERO mat)
                nombre=nom
                semestre=sem
                cantMaterias=mat
                promCredito=0
          FIN_Estudiante
          PUBLICO VOID calcularPromedio()
                Materia mat
                REAL suma=0
                ENTERO k, cre
                REAL no
                ENTERO sumaCreditos=0
                PARA (k=1; k<=cantMaterias; k=k+1) HAGA
                ESCRIBA: "DIGITE LA NOTA:"
                LEA: no
                ESCRIBA: "DIGITE NÚMERO DE CRÉDITOS:"
                LEA: cre
                mat=Materia (no, cre);
                suma=suma+mat.nota*mat.creditos
                sumaCreditos=sumaCreditos+mat.creditos
               FIN_PARA
              promCredito=suma/sumaCreditos
          FIN_calcularPromedio

    PUBLICO VOID mostrarDatos ()
                    ESCRIBA: "EL NOMBRE DEL ESTUDIANTE ES:", nombre
                    ESCRIBA: "EL SEMESTRE ACTUAL ES:", semestre
                    ESCRIBA: "EL PROMEDIO CRÉDITO ES:", promCredito
          FIN_mostrarDatos

      METODO PRINCIPAL ()
              Estudiante obj
              CADENA nom
              ENTERO sem, mat
              ESCRIBA: "DIGITE EL NOMBRE:"
```

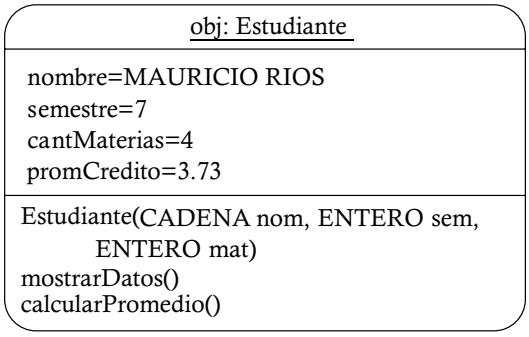
```
            LEA: nom
            ESCRIBA: "DIGITE EL SEMESTRE"
            LEA: sem
            ESCRIBA: "DIGITE LA CANTIDAD DE MATERIAS"
            LEA: mat
            obj=Estudiante (nom, sem, mat)
            obj.calcularPromedio()
           obj.mostrardatos()
  FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Suponiendo que el estudiante Mauricio Ríos de séptimo semestre tiene cuatro materias, en la ejecución del método calcularPromedio() se ingresarán los siguientes valores para estas:



Al finalizar la ejecución del método principal se tendrá un objeto de la clase Estudiante con la siguiente información:



Salida

```
EL NOMBRE DEL ESTUDIANTE ES: MAURICIO RÍOS
EL SEMESTRE ACTUAL ES: 7
EL PROMEDIO CREDITO ES: 3.73
```

7.11 Métodos accesores

Para proteger la información de un objeto se sugiere declarar las variables miembro de la clase como privadas y crear métodos accesores que permitan consultar y modificar el valor de las estas. De esta manera, si otra clase desea acceder las variables privadas, entonces invoca a los métodos accesores. Para aplicar esta sugerencia se deben crear dos métodos para cada variable miembro de la clase:

- Modificar variable (setVariable): Permite cambiar el valor de una variable. El nuevo valor es enviado como parámetro de entrada al método y no retorna ningún valor.
- Obtener variable (getVariable): Permite consultar el valor asignado a una variable de la clase, no recibe parámetros de entrada y retorna el valor que se desea consultar.

Siguiendo esta sugerencia, si una clase tiene n variables miembro, entonces tendrá 2n métodos accesores.

Ejercicio resuelto Cap7-Ejer8
Adicionar los métodos accesores a la clase Persona codificada en el Ejercicio resuelto Cap7-Ejer4.

Algoritmo

```
CLASE Persona
    PRIVADO CADENA nombre
    PRIVADO ENTERO añoNacimiento
    PRIVADO CARACTER sexo
    PRIVADO REAL peso
    PRIVADO REAL estatura
    PUBLICO Persona (CADENA nom, ENTERO an, CARACTER sex, REAL
    pe, REAL est)
                nombre=nom
                añoNacimiento=an
                sexo=sex
                peso=pe
                estatura=est
    FIN_persona
    PUBLICO CADENA getNombre()
      RETORNE nombre
```

```
    FIN_getNombre
    PUBLICO ENTERO getAñoNacimiento()
      RETORNE nombre
    FIN_getAñoNacimiento
    PUBLICO CARACTER getSexo()
      RETORNE sexo
    FIN_getSexo
    PUBLICO REAL getPeso()
     RETORNE peso
    FIN_getPeso
    PUBLICO REAL getEstatura()
     RETORNE estatura
    FIN_getEstatura
    PUBLICO VOID setNombre(CADENA nom)
       nombre=nom
    FIN_setNombre
    PUBLICO VOID setAñoNacimiento(ENTERO an)
      añoNacimiento=an
    FIN_AñoNacimiento
    PUBLICO VOID setSexo(CARACTER sex)
       sexo=sex
    FIN_setSexo
    PUBLICO VOID setPeso(REAL p)
      peso=p
    FIN_setPeso
    PUBLICO VOID setEstatura(REAL est)
      estatura=est
    FIN_setEstatura
    PUBLICO VOID CalcularMasaCorporal()
              indice=peso/estatura^2
              ESCRIBA: "EL ÍNDICE ES:", indice
    FIN_CalcularMasaCorporal
    PUBLICO VOID CalcularEdad ()
              edad=2014-añoNacimiento
              ESCRIBA: "LA EDAD ES:", edad
    FIN_CalcularEdad
    ~Persona ()
              ESCRIBA: "DESTRUYENDO EL OBJETO"
    FIN_~Persona
FIN_CLASE

METODO PRINCIPAL ()
      Persona objeto1, objeto2
      objeto1=Persona ("SAMUEL", 2010, "H", 13.7, 90.5)
      objeto2=Persona ("MARÍA ISABEL", 2002 , "M", 45, 140)
      objeto1.setNombre ("DAVID")
```

```
        ESCRIBA: "EL SEXO ES:", objeto2.getSexo()
        objeto2.calcularEdad()
    FIN_PRINCIPAL
FIN_CLASE
```

Ejercicio resuelto Cap7-Ejer9

Construir un programa que calcule las estadísticas finales de un curso, ingresando el nombre de este y la cantidad de estudiantes que tiene matriculados. El programa debe permitir el ingreso de la nota final y del semestre que cursa cada estudiante para informar la nota promedio del curso, la cantidad de estudiantes que ganaron, la cantidad de estudiantes que perdieron la materia y el semestre promedio en el que se encuentran los estudiantes.

1. Identificación de datos, acciones y limitaciones

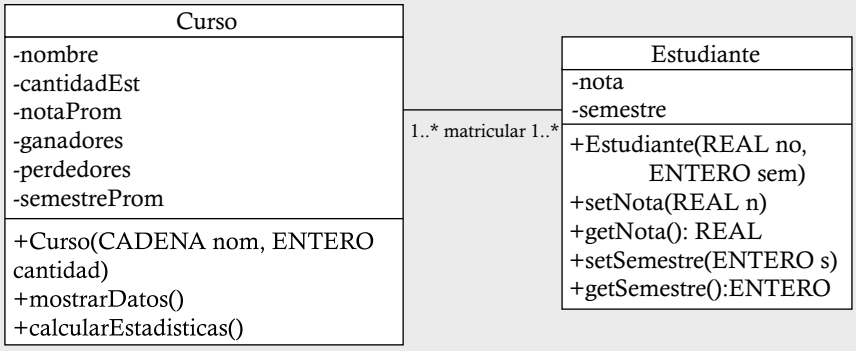
- Datos de entrada: nombre del curso, cantidad de estudiantes del curso, nota de cada estudiante, semestre que cursa cada estudiante.
- Datos de salida: nota promedio del curso, cantidad de estudiantes que ganaron, cantidad de estudiantes que perdieron, semestre promedio que cursan los estudiantes.
- Acciones: calcular las estadísticas del curso.
- Limitaciones: No se ingresan los nombres de los estudiantes porque no se necesitan.

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): estadística, curso, estudiante.
- Relación de los sustantivos con los datos y las acciones:
  Estadística: N/A.
  Curso: nombre, cantidad de estudiantes, cálculo de estadísticas.
  Estudiante: nota, semestre que cursa.
- Clases seleccionadas: Curso y Estudiante.
- Definición de constructores y destructores para cada clase:
  Curso (CADENA nom, ENTERO cantidad): Este constructor recibe como parámetros algunos datos de la clase.
  Estudiante (REAL no, ENTERO sem): Este constructor recibe como parámetros todos los datos de la clase.

Continúa en la siguiente página

- Diagramación de clases con sus variables y métodos: En ambas clases las variables son declaradas como privadas para proteger la información. En la clase Estudiante se deben crear métodos accesores para poder hacer los cálculos en la clase Curso. En esta no es necesario crear métodos accesores ya que no existe ninguna otra clase que requiera acceder a las variables de ella. Véase que los métodos de la clase Curso pueden acceder sin ningún problema las variables privadas de la clase. La relación entre las clases es representada por medio de una asociación que se lee de la siguiente forma:
  - Un curso activo tiene matriculados uno o muchos estudiantes.
  - Un estudiante activo tiene matriculados uno o muchos cursos.



- **Explicación de los métodos de la clase Curso:**
  Curso: Constructor que recibe algunas variables de la clase. Inicializa las otras variables.
  mostrarDatos: Método que muestra por pantalla las variables de la clase.
  calcularEstadisticas: Método que calcula los datos solicitados en este ejercicio.
- **Explicación de los métodos de la clase Estudiante:**
  Estudiante: Constructor de la clase que recibe los valores por asignar a las variables de la clase.
  setNota y getNota: Métodos accesores de la variable nota.
  setSemestre y getSemestre: Métodos accesores de la variable semestre.

Continúa en la siguiente página

3. Definición del método principal
Se debe ingresar el nombre del curso y la cantidad de estudiantes para enviárselos al constructor de la clase Curso. Después de instanciar el objeto de la clase Curso, se debe invocar el método que se encarga de calcular las estadísticas. No es necesario instanciar objetos de la clase Estudiante ya que estos objetos se crean dentro de la clase Curso.

Definición de variables:
obj: Objeto de la clase Curso desde el cual se invocan los métodos de la clase.
nombreCurso: Variable que almacena el nombre del curso.
numeroEstudiantes: Variable que almacena la cantidad de estudiantes del curso.

Algoritmo

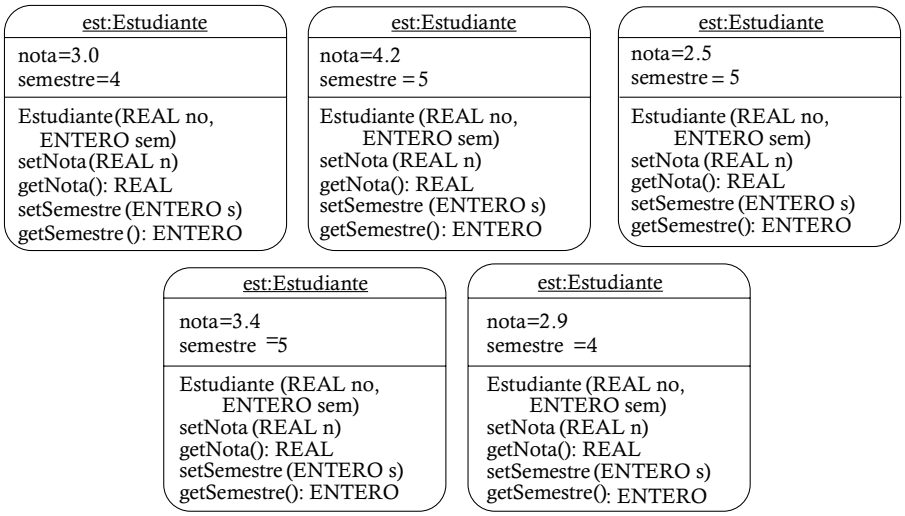
```
PUBLICA CLASE Estudiante
    PRIVADO REAL nota
    PRIVADO ENTERO semestre

    PUBLICO Estudiante(REAL no, ENTERO sem)
       nota=no
       semestre=sem
    FIN_Estudiante
    PUBLICO VOID setNota(REAL n)
    nota=n
    FIN_setNota
    PUBLICO REAL getNota()
         RETORNE nota
    FIN_getNota
    PUBLICO VOID setSemestre(ENTERO s)
         semestre=s
    FIN_getSemestre
    PUBLICO ENTERO getSemestre()
         RETORNE semestre
    FIN_setSemestre
FIN_CLASE

PUBLICA CLASE Curso
    PRIVADO CADENA nombre
    PRIVADO ENTERO cantidadEst
    PRIVADO ENTERO perdedores
    PRIVADO ENTERO ganadores
```

```
PRIVADO REAL notaProm
PRIVADO ENTERO semestreProm
PUBLICO Curso(CADENA nom, ENTERO cantidad)
    nombre=nom
    cantidadEst=cantidad
    perdedores=0
    ganadores=0
    notaProm=0
    semestreProm=0
FIN_curso
PUBLICO VOID mostrarDatos ()
    ESCRIBA: "EL NOMBRE DEL CURSO ES:", nombre
    ESCRIBA: "CANTIDAD DE ESTUDIANTES:",cantidadEst
    ESCRIBA: "LA NOTA PROMEDIO DEL CURSO ES:", notaProm
    ESCRIBA: "ESTUDIANTES QUE GANARON:", ganadores
    ESCRIBA: "ESTUDIANTES QUE PERDIERON:", perdedores
    ESCRIBA: "EL SEMESTRE PROMEDIO ES:", semestreProm
FIN_mostrarDatos
PUBLICO VOID calcularEstadisticas()
    Estudiante est
    REAL sumaNota=0, no
    ENTERO sumaSemestre=0
    ENTERO k, sem
    PARA (k=1; k<=cantidadEst; k=k+1) HAGA
            ESCRIBA: "DIGITE LA NOTA:"
            LEA: no
            ESCRIBA: "DIGITE EL SEMESTRE:"
            LEA: sem
                est=Estudiante(no, sem)
                sumaNota=sumaNota+est.getNota()
                SI (est.getNota()<3)
                        perdedores=perdedores+1
                SI_NO
                        ganadores=ganadores+1
                FIN_SI
                sumaSemestre=sumaSemestre+est.getSemestre()
    FIN_PARA
        notaProm=sumaNota/cantidadEst
        semestreProm=sumaSemestre/cantidadEst
        mostrarDatos()
FIN_calcularEstadisticas
METODO PRINCIPAL ()
   Curso obj
   CADENA nombreCurso
   ENTERO numeroEstudiantes
   ESCRIBA: "DIGITE EL NOMBRE DEL CURSO"
```

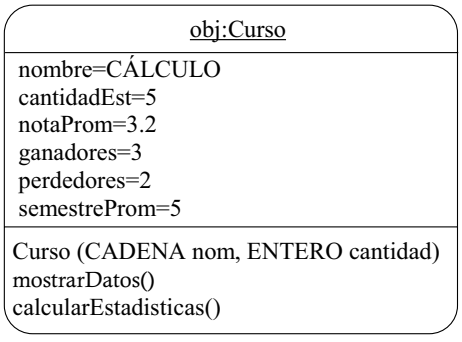
```
            LEA: nombreCurso
            ESCRIBA: "DIGITE LA CANTIDAD DE ESTUDIANTES DEL
            CURSO"
            LEA:numeroEstudiantes
            obj=Curso(nombreCurso, numeroEstudiantes)
            obj.calcularEstadisticas()
        FIN_PRINCIPAL
    FIN_CLASE
```

Prueba de escritorio

Suponiendo como entrada el curso Cálculo con cinco estudiantes, en la ejecución del método calcularEstadisticas() se ingresarán los siguientes estudiantes:



Al finalizar la ejecución del método principal se tendrá un objeto de la clase Curso con la siguiente información:



Salida

```
EL NOMBRE DEL CURSO ES: CÁLCULO
CANTIDAD DE ESTUDIANTES: 5
LA NOTA PROMEDIO DEL CURSO ES: 3.2
ESTUDIANTES QUE GANARON: 3
ESTUDIANTES QUE PERDIERON: 2
EL SEMESTRE PROMEDIO ES: 5
```

7.12 Sobrecarga de métodos

La sobrecarga permite a una misma clase tener varios métodos con el mismo nombre, inclusive con el mismo valor de retorno, pero con diferentes parámetros. De este modo se puede redefinir un mismo método las veces que sea necesario dentro de una misma clase variando la cantidad de parámetros o el tipo de dato de estos.

Al ejecutarse, se identifica que se invoca uno u otro método según la cantidad y el tipo de los parámetros. Es muy importante establecer que dos métodos no pueden distinguirse solamente por el tipo de parámetro de retorno.

Esta propiedad ofrece una variación muy importante a las clases permitiendo tener una sobrecarga de constructores. De esta manera se puede inicializar un objeto de diferentes formas: ingresando los datos como parámetros, solicitando la información dentro del constructor, inicializando solo algunas de las variables, etcétera.

Ejercicio resuelto Cap7-Ejer10
Desarrollar un conversor de cambio monetario de peso a dólar y de peso a euro. El usuario debe tener la posibilidad de seleccionar si desea ingresar el valor de las tasas de cambio o si desea utilizar las que tenga el programa por defecto. Además, se debe tener la posibilidad de convertir diferentes cantidades.

1. Identificación de datos, acciones y limitaciones
- Datos de entrada: tasas de cambio y valores por convertir.
- Datos de salida: valor en dólares y valor en euros.
- Acciones: convertir las cantidades.
- Limitaciones: La conversión es solo de peso a dólar y de peso a euro.

Continúa en la siguiente página

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): conversor, usuario.
- Relación de los sustantivos con los datos y las acciones:
  Conversor: tasa de cambio, valores por convertir, valor en dólares, valor en euros.
  Usuario: N/A.
- Clase seleccionada: Conversor.
- Definición de constructores y destructores para la clase: El problema requiere dos posibilidades para inicializar las variables miembro de la clase; por lo tanto se crean dos constructores para la clase:
  Conversor (REAL dólar, REAL euro): Este constructor recibe como parámetros las tasas de cambio.
- Conversor (): Este constructor inicializa las tasas de cambio con valores por defecto.
- Diagramación de la clase con sus variables y métodos.

Conversor
-TRMDolar -TRMEuro
+Conversor (REAL dólar, REAL euro) +Conversor() +convertirPesoDolar(REAL valor) +convertirPesoEuro(REAL valor)

Explicación de los métodos:

Conversor con parámetros: Constructor que recibe los valores para inicializar las variables de la clase.

Conversor sin parámetros: Constructor inicializa las variables de la clase con valores por defecto.

convertirPesoDolar: Método que convierte un valor en pesos a dólar.

convertirPesoEuro: Método que convierte un valor en pesos a euros.

3. Definición del método principal

El usuario inicialmente debe indicar si desea ingresar las tasas de cambio o si prefiere utilizar las que el programa trae por defecto. Esto se realiza por medio de una pregunta que conlleva a invocar cualquiera de los dos constructores de la clase Conversor. Una vez se ha creado el objeto de la

Continúa en la siguiente página

clase Conversor se procede a solicitar el valor en pesos que se desea convertir invocando los métodos de la clase. Esta solicitud es implementada dentro de un ciclo que permite al usuario convertir diferentes cantidades.

Definición de variables:
obj: Objeto de la clase Conversor desde la cual se invocan los métodos de la clase.
dolar, euro: Variables para almacenar la tasa de cambio del dólar y del euro, en caso de que el usuario desee ingresar estos valores.
valor: Variable que almacena el valor que se desea convertir.
resultado: Variable que almacena el resultado de la conversión.
opción: Variable que almacena la decisión del usuario sobre si desea ingresar o no las tasas de cambio.

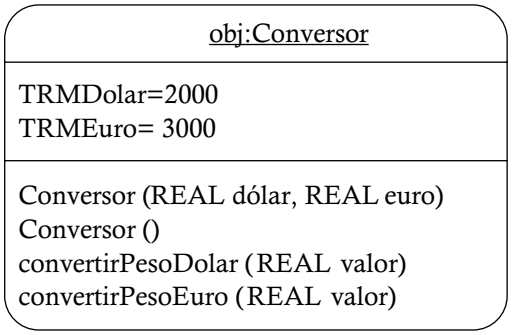
Algoritmo

```
CLASE Conversor
    PRIVADO REAL TRMDolar
    PRIVADO REAL TRMEuro
    PUBLICO Conversor ()
            TRMDolar=2000
            TRMEuro=3000
    FIN_Conversor
    PUBLICO Conversor (REAL dolar, REAL euro)
            TRMDolar=dolar
            TRMEuro=euro
    FIN_Conversor
    PUBLICO REAL convertirPesoDolar (REAL pesos)
                    REAL conversion=pesos/TRMDolar
                    RETORNE conversion
    FIN_convertirPesoDolar
    PUBLICO REAL convertirPesoEuro (REAL pesos)
                    REAL conversion=pesos/TRMEuro
                    RETORNE conversion
    FIN_convertirPesoEuro
    METODO PRINCIPAL ()
        Conversor obj
    REAL dolar, euro, valor, resultado
        CADENA opcion
          ESCRIBA: "DIGITE SI PARA INGRESAR LAS TASAS DE
          CAMBIO O  DIGITE NO PARA UTILIZAR VALORES POR
          DEFECTO"
          LEA: opcion
```

```
            SI (opcion==“SI”) ENTONCES
                ESCRIBA: “DIGITE EL VALOR DEL DÓLAR:”
                    LEA: dolar
                ESCRIBA: “DIGITE EL VALOR DEL EURO:”
                    LEA: euro
                    obj= Conversor (dolar, euro)
            SI_NO
                    obj= Conversor()
            FIN_SI
            HAGA
                    ESCRIBA: “DIGITE EL VALOR EN PESOS POR
                    CONVERTIR:”
                    LEA: valor
                    resultado= obj.convertirPesoDolar (valor)
                    ESCRIBA: “VALOR EN DÓLARES:”, resultado
                    resultado= obj.convertirPesoEuro (valor)
                    ESCRIBA: “VALOR EN EUROS:”, resultado
                    ESCRIBA: “¿DESEA CONVERTIR UN NUEVO
                    VALOR? SI/NO”
                    LEA: opcion
            MIENTRAS  (opcion==“SI”)
        FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Suponiendo que el usuario selecciona utilizar los valores por defecto para las tasas de cambio, se crea un objeto con las siguientes características:



Salida
Si el usuario ingresa la cantidad para convertir de $1.000.000, entonces la salida del programa es:

```
VALOR EN DÓLARES: 500
VALOR EN EUROS: 333.33
```

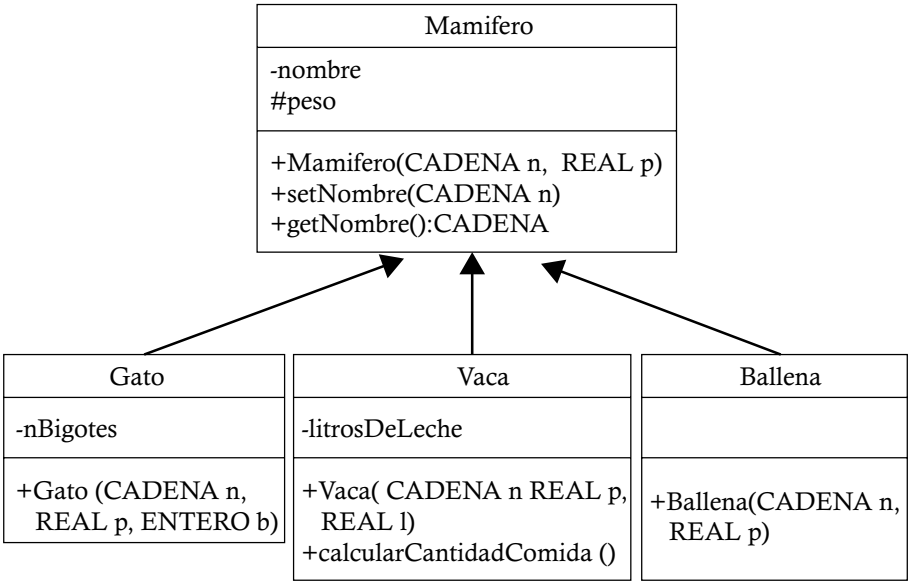
7.13 Herencia y reutilización

Al igual que los hijos heredamos características de nuestros padres, en la POO podemos obtener clases hijas con el mismo contenido de su clase padre. De esta manera se obtiene una reutilización de las clases, la cual es una de las propiedades más deseadas en la programación ya que conlleva un ahorro de tiempo y de líneas de código.

La reutilización ha sido abordada en la programación estructural por medio de funciones. Esto permite crear líneas de código que pueden ser invocadas tantas veces como sea necesario. En la POO este concepto es extendido a las clases, de manera que no solo se utilizan clases existentes, sino que también se adicionan nuevas variables y métodos a estas sin modificar el contenido original. Esta reutilización se logra por medio de la herencia de clases, donde la clase hija tiene a su disposición todos los miembros de la clase padre, llamada también superclase o clase base. La clase hija es llamada también subclase o clase heredada.

Gráficamente, en el diagrama de clases la herencia se representa con una flecha desde la subclase hacia la superclase, que indica que la subclase hereda los métodos y atributos de la superclase. De esta manera, en la subclase se pueden manipular los atributos y métodos visibles de la clase padre, es decir, los que fueron declarados públicos y protegidos. Los métodos y atributos privados solo pueden ser accedidos desde los métodos visibles de la superclase.

En la figura siguiente la clase Mamifero es la clase padre, y las clases Gato, Vaca y Ballena heredan de ella, obteniendo las variables y métodos visibles de la clase padre, es decir que pueden acceder a la variable peso y a los tres métodos públicos definidos en la clase padre. Por el contrario, la variable nombre no puede ser accedida directamente en las clases hijas; solo pueden manipularse a través de los métodos accesores que son públicos (setNombre y getNombre).



En la figura vemos además que a las clases hijas pueden adicionar variables y métodos propios de cada animal sin afectar el comportamiento de las otras clases que heredaron.

El pseudocódigo de la clase base se presenta a continuación:

```
CLASE Mamifero
      PRIVADO CADENA nombre
      PROTEGIDO REAL peso
      PUBLICO Mamifero (CADENA n, REAL p)
            nombre=n
            peso=p
      FIN_Mamifero
      PUBLICO VOID setNombre (CADENA n)
            nombre= n
      FIN_setNombre
      PUBLICO CADENA getNombre()
            retorne nombre
      FIN_getNombre
FIN_CLASE
```

La clase heredada Gato declara un constructor que recibe las dos variables heredadas y una nueva variable propia de la clase hija. Para inicializar las variables heredadas, se invoca el constructor de la clase padre:

```
CLASE Gato HEREDA CLASE Mamifero
    PRIVADO ENTERO nBigotes
    PUBLICO Gato (CADENA n, REAL p, ENTERO b)
        Mamifero(n,p)
        nBigotes=b
    FIN_Gato
FIN_CLASE
```

La clase heredada Vaca, además del constructor, tiene un método para calcular la cantidad de comida que su dueño debe suministrarle para que produzca constantemente los litros de leche deseados. Nótese que la variable peso puede ser utilizada directamente por el método ya que es una variable heredada de la clase Mamífero:

```
CLASE Vaca HEREDA CLASE Mamifero
    PRIVADO REAL litrosDeLeche
    PUBLICO Vaca (CADENA n, REAL p, REAL l)
        Mamifero(n,p)
        litrosDeLeche=l
    FIN_Vaca
    PUBLICO VOID calcularCantidadComida ()
        REAL kilosComida=peso/litrosDeLeche
        ESCRIBA: "LA CANTIDAD DE COMIDA ES:",  kilosComida
    FIN_calcularCantidadComida
FIN_CLASE
```

Finalmente, la clase heredada Ballena no tiene variables adicionales a las ya heredadas de su clase padre:

```
CLASE Ballena HEREDA CLASE Mamifero
    PUBLICO Balena (CADENA n, REAL p)
        Mamifero(n,p)
    FIN_Ballena
FIN_CLASE
```

Ejercicio resuelto Cap7-Ejer11

Crear el método principal que permita instanciar objetos de las clases Gato, Vaca y Ballena definidas anteriormente, invocando sus métodos y mostrando el contenido de los objetos.

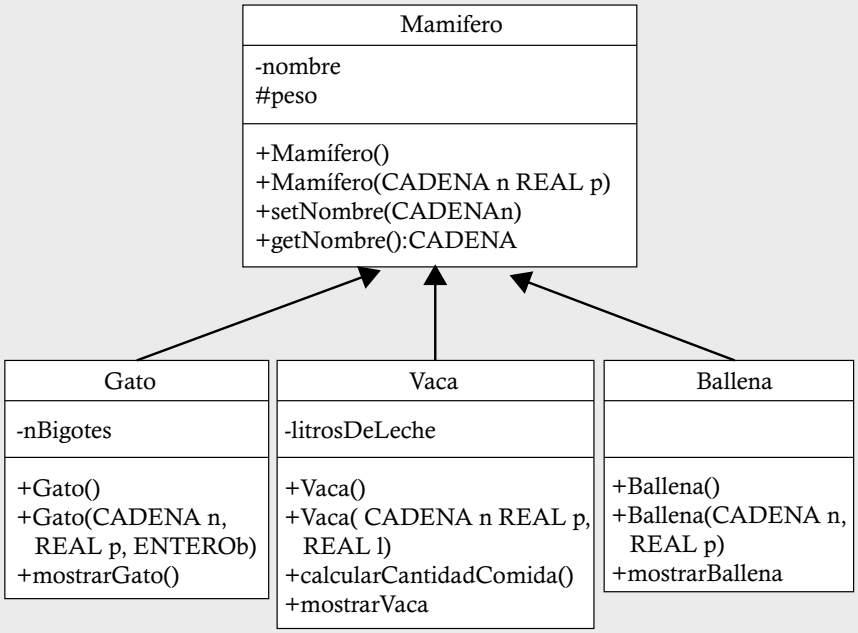
Continúa en la siguiente página

1. Identificación de datos, acciones y limitaciones

- Datos de entrada: peso (gato, vaca y ballena), nombre (gato, vaca y ballena), número de bigotes (gato), litros de leche (vaca).
- Datos de salida: peso (gato, vaca y ballena), nombre (gato, vaca y ballena), número de bigotes (gato), litros de leche (vaca) y cantidad de comida (vaca).
- Acciones: capturar los datos, mostrarlos y calcular la cantidad de comida de la vaca.
- Limitaciones: Solo se va a crear un objeto de cada clase.

2. Definición de clases

Además de las clases Mamífero, Gato, Vaca y Ballena presentadas anteriormente, se creará una clase con el método principal donde se realizarán las acciones solicitadas. Para acceder a las variables privadas y protegidas de las clases ya creadas es necesario adicionar un método que muestre los datos de cada clase. Adicionalmente, los constructores se presentan sobrecargados para ofrecer diferentes formas de solicitar los datos.



Continúa en la siguiente página

- **Explicación de los métodos de la clase Mamífero:**
  Mamífero con parámetros y sin parámetros: Constructores de la clase.
  setNombre y getNombre: Métodos accesores de la variable nombre.
- **Explicación de los métodos de la clase Gato:**
  Gato con parámetros y sin parámetros: Constructores de la clase.
  mostrarGato: Método que muestra por pantalla el contenido de las variables de la clase.
- **Explicación de los métodos de la clase Vaca:**
  Vaca con parámetros y sin parámetros: Constructores de la clase.
  calcularCantidadComida: Método que calcula la cantidad de comida que se le debe dar a la vaca.
  mostrarVaca: Método que muestra por pantalla el contenido de las variables de la clase.
- **Explicación de los métodos de la clase Ballena:**
  Ballena con parámetros y sin parámetros: Constructores de la clase.
  mostrarBallena: Método que muestra por pantalla el contenido de las variables de la clase.

3. Definición del método principal

En este método se solicitan los datos de cada clase, se instancian los objetos y se invocan sus métodos.

Definición de variables:

obj1: Variable que almacena un objeto de tipo Gato.
obj2: Variable que almacena un objeto de tipo Vaca.
obj3: Variable que almacena un objeto de tipo Ballena.
nom: Variable que almacena el nombre de los animales.
p: Variable que almacena el peso de los animales.
nb: Variable que almacena la cantidad de bigotes del gato.

Algoritmo

```
CLASE Mamifero
    PRIVADO CADENA nombre
    PROTEGIDO REAL peso
    PUBLICO Mamifero()
            ESCRIBA: "INGRESE EL NOMBRE:"
            LEA: nombre
            ESCRIBA: "INGRESE EL PESO:"
```

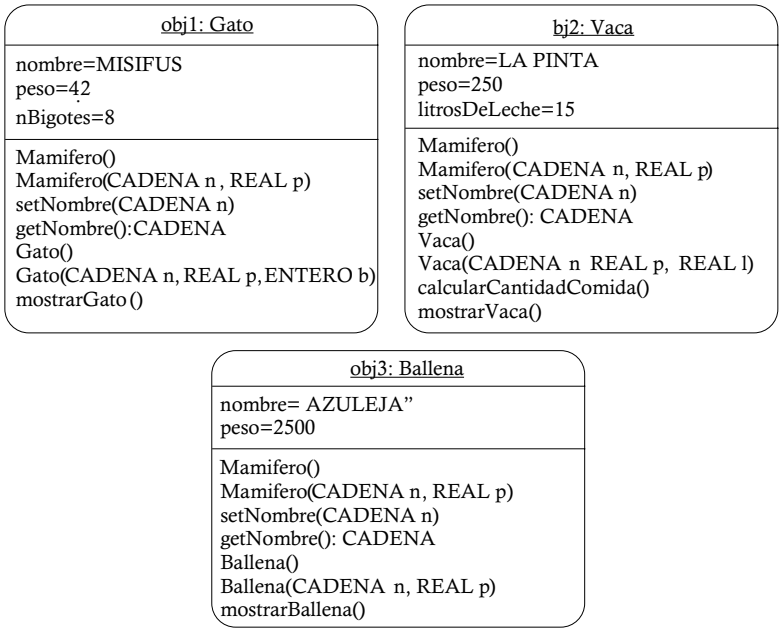
```
            LEA: peso
      FIN_Mamifero
      PUBLICO Mamifero(CADENA n, REAL p)
            nombre=n
            peso=p
      FIN_Mamifero
      PUBLICO VOID setNombre(CADENA n)
            nombre=n
      FIN_setNombre
      PUBLICO CADENA getNombre()
            RETORNE nombre
      FIN_getNombre
FIN_CLASE

CLASE Gato HEREDA CLASE Mamifero
      PRIVADO ENTERO nBigotes
      PUBLICO Gato()
      Mamifero()
            ESCRIBA: "INGRESE LA CANTIDAD DE BIGOTES:"
            LEA: nBigotes
      FIN_Gato
      PUBLICO Gato(CADENA n, REAL p, ENTERO b)
            Mamifero(n,p)
            nBigotes=b
      FIN_Gato

PUBLICO VOID mostrarGato()
            ESCRIBA: "CLASE GATO:"
            ESCRIBA: "NOMBRE:", getNombre()
            ESCRIBA: "PESO:", peso
            ESCRIBA: "NÚMERO DE BIGOTES:", nBigotes
      FIN_mostrarGato
FIN_CLASE

CLASE Vaca HEREDA CLASE Mamifero
      PRIVADO REAL litrosDeLeche
      PUBLICO Vaca()
      Mamifero()
            ESCRIBA: "INGRESE LOS LITROS DE LECHE:"
            LEA: litrosDeLeche
      FIN_Vaca
      PUBLICO Vaca(CADENA n, REAL p, REAL l)
            Mamifero(n,p)
            litrosDeLeche = l
      FIN_Vaca
      PUBLICO VOID calcularCantidadComida()
```

```
            REAL kilosComida=peso/(2*litrosDeLeche)
            ESCRIBA: "LA CANTIDAD DE COMIDA ES:",  kilosComida
    FIN_calcularCantidadComida
    PUBLICO VOID mostrarVaca()
            ESCRIBA: "CLASE VACA:"
            ESCRIBA: "NOMBRE:", getNombre()
            ESCRIBA: "PESO:", peso
            ESCRIBA: "LITROS DE LECHE:", litrosDeLeche
    FIN_mostrarVaca
FIN_CLASE

CLASE Ballena HEREDA CLASE Mamifero
    PUBLICO Ballena()
    Mamifero()
    FIN_Ballena
    PUBLICO Ballena(CADENA n, REAL p)
            Mamifero(n,p)
    FIN_Ballena
    PUBLICO VOID mostrarBallema()
            ESCRIBA: "CLASE BALLENA"
            ESCRIBA: "NOMBRE:", getNombre()
            ESCRIBA: "PESO:", peso
    FIN_mostrarBallena
FIN_CLASE

CLASE PRINCIPAL
    METODO PRINCIPAL()
        CADENA nom
        REAL p
        ENTERO nb
        ESCRIBA: "CREANDO UN GATO"
        ESCRIBA: "DIGITE EL NOMBRE:"
        LEA: nom
        ESCRIBA: "DIGITE EL PESO:"
        LEA: p
        ESCRIBA: "DIGITE EL NÚMERO DE BIGOTES:"
        LEA: nb
        Gato obj1=Gato(nom, p, nb)
            ESCRIBA: "CREANDO UNA VACA"
            Vaca obj2= Vaca()
            ESCRIBA: "CREANDO UNA BALLENA"
            ESCRIBA: "INGRESE EL NOMBRE:"
            LEA: nom
            ESCRIBA: "INGRESE EL PESO:"
            LEA: p
            Ballena obj3=Ballena(nom, p)
```

```
            obj1.mostrarGato()
            obj2.mostrarVaca()
            obj2.calcularCantidadComida()
            obj3.mostrarBallena()
        FIN_PRINCIPAL
FIN_CLASE
```

Prueba de Escritorio

Suponiendo que el usuario ingresará los siguientes valores para los tres objetos creados, se tiene:



Nótese que cada objeto tiene a su disposición las variables y los métodos de la superclase Mamífero como propios.

Salida

Al ejecutarse los métodos de mostrar datos de cada clase, se obtiene la siguiente salida:

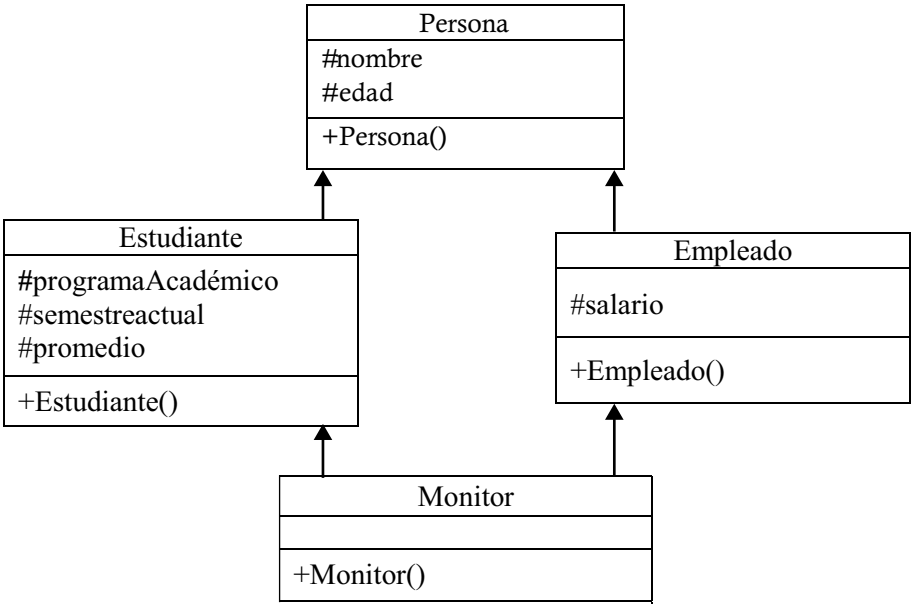
```
CLASE GATO
NOMBRE: MISIFUS
PESO: 4.2
NÚMERO DE BIGOTES: 8

CLASE VACA
NOMBRE: LA PINTA
PESO: 250
LITROS DE LECHE: 15
LA CANTIDAD DE COMIDA ES: 8.33

CLASE BALLENA
NOMBRE: AZULEJA
PESO: 2500
```

7.13.1 Tipos de herencia

Las clases pueden tener una herencia simple o compuesta. En la herencia simple una subclase puede heredar variables y métodos de una única superclase, mientras que en la herencia compuesta una subclase puede heredar variables y métodos de varias superclases, es decir que puede tener varias clases padre.

La herencia simple es el caso más común de herencia y es soportada por todos los lenguajes de programación orientados a objetos. Sin embargo, la herencia compuesta no es soportada por todos los lenguajes dado que se pueden tener incongruencias si se replican las variables o los métodos en las clases heredadas.

En el siguiente ejemplo la clase Monitor tiene replicadas las variables nombre y edad, que hereda por sus dos padres: Estudiante y Empleado.



Ejercicio resuelto Cap7-Ejer12

Elaborar un programa para controlar la cantidad de personas que ingresan a la universidad utilizando algún medio de transporte diferente al público. Para esto se contabiliza la cantidad de vehículos y motos que ingresan en un día a la institución. De cada medio de transporte se almacena su capacidad de pasajeros y la cantidad que ingresan en un día. Adicionalmente, se almacena cuántos vehículos son a gas y cuántos son a gasolina. De las motos se desea almacenar cuántas tienen un cilindraje mayor a 250 cc. Se desea calcular la cantidad máxima de personas que pueden ingresar a la institución empleando alguno de los medios de transporte mencionados.

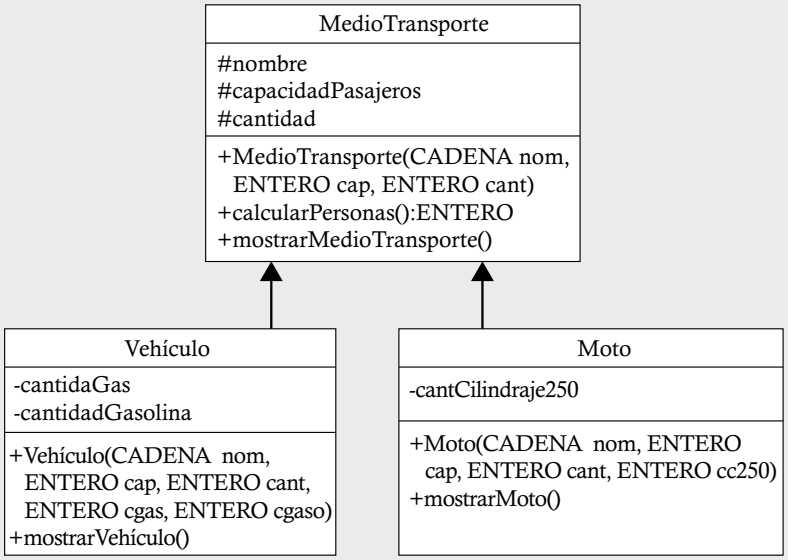
1. Identificación de datos, acciones y limitaciones

- Datos de entrada: capacidad de pasajeros, cantidad de pasajeros, cantidad de vehículos a gas y a gasolina, cantidad de motos con cilindraje mayor a 250 cc.
- Datos de salida: cantidad máxima de personas
- Acciones: capturar los datos, mostrarlos y calcular la cantidad de personas que ingresan en cada medio de transporte.
- Limitaciones: Solo se van a contabilizar las personas de vehículos y motos.

Continúa en la siguiente página

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): personas, medio de transporte, vehículos, motos.
- Relación de los sustantivos con los datos y las acciones:
  Personas: N/A.
  Medio de transporte: capacidad de pasajeros, cantidad, calcular cantidad máxima de personas.
  Vehículo: cantidad de vehículos a gas y a gasolina.
  Moto: cantidad de motos con cilindraje mayor a 250 cc.
- Clases seleccionadas: MedioTransporte, Vehiculo y Moto.
- Definición de constructores y destructores: Cada clase tiene un constructor que inicializa las variables miembro.
- Diagramación de las clases con sus variables y métodos: las clases Vehiculo y Moto heredan de la clase MedioTransporte.



- **Explicación de los métodos de la clase MedioTransporte:**
  MedioTransporte: Constructor de la clase.
  calcularPersonas: Método que calcula la cantidad de personas que se movilizan por cada medio de transporte.
  mostrarMedioTransporte: Método que muestra por pantalla las variables de la clase.

Continúa en la siguiente página

- **Explicación de los métodos de la clase Vehiculo:**
  Vehiculo: Constructor de la clase.
  mostrarVehiculo: Método que muestra por pantalla las variables de la clase.

- **Explicación de los métodos de la clase Moto:**
  Moto: Constructor de la clase.
  mostrarMoto: Método que muestra por pantalla las variables de la clase.

3. Definición del método principal

En este método se solicitan los datos de cada clase, se instancian los objetos y se invocan sus métodos. Este método es incluido en una clase diferente a las diagramadas anteriormente.

Definición de variables:

obj1: Variable que almacena un objeto de la clase Vehiculo.
obj2: Variable que almacena un objeto de la clase Moto.
cant: Variable que almacena la cantidad de vehículos y de motos.
cantGas y cantGaso: Variable que almacena la cantidad de vehículos a gas y a gasolina.
cantC250: Variable que almacena la cantidad de motos con cilindraje mayor a 250 cc.

Algoritmo

```
CLASE MedioTransporte
   PROTEGIDO CADENA nombre
   PROTEGIDO ENTERO capacidadPasajeros
   PROTEGIDO ENTERO cantidad
   PUBLICO MedioTransporte(CADENA  nom, ENTERO cap, ENTERO cant)
            nombre=nom
            capacidadPasajeros=cap
            cantidad=cant
     FIN_MedioTransporte

     PUBLICO ENTERO calcularPersonas( )
            RETORNE (capacidadPasajeros*cantidad)
     FIN_calcularPersonas
```

```
        PUBLICO VOID mostrarMedioTransporte()
                ESCRIBA: "NOMBRE:", nombre
                ESCRIBA: "CAPACIDAD DE PASAJEROS:", capacidadPasajeros
                ESCRIBA: "CANTIDAD QUE INGRESAN/DÍA:", cantidad
        FIN_mostrarMedioTransporte
FIN_CLASE

CLASE Vehiculo HEREDA CLASE MedioTransporte
        PRIVADO ENTERO cantidadGas
        PRIVADO ENTERO cantidadGasolina
        PUBLICO Vehiculo(CADENA  nom, ENTERO cap, ENTERO cant,
                                ENTERO cgas, ENTERO cgaso)
        MedioTransporte(nom, cap, cant)
                cantidaGas=cgas
                cantidaGasolina=cgaso
        FIN_Vehiculo
        PUBLICO VOID mostrarVehiculo()
                mostrarMedioTransporte()
                ESCRIBA: "CANTIDAD DE VEHÍCULOS A GAS:",
                cantidadGas
                ESCRIBA: "CANTIDAD DE VEHÍCULOS A GASOLINA:",
                cantidadGasolina
        FIN_mostrarVehículo
FIN_CLASE

CLASE Moto HEREDA CLASE MedioTransporte
        PRIVADO ENTERO cantCilindraje250
                PUBLICO Moto(CADENA  nom, ENTERO  cap, ENTERO cant,
                ENTERO cc250)
        MedioTransporte(nom, cap, cant)
                  cantCilindraje250=cc250
        FIN_Moto
        PUBLICO VOID mostrarMoto()
           mostrarMedioTransporte()
           ESCRIBA: "CANTIDAD DE MOTOS CON CILINDRAJE MAYOR
           A 250:",
           cantCilindraje250
        FIN_mostrarMoto
FIN_CLASE
CLASE PRINCIPAL
        METODO PRINCIPAL()
           ENTERO cant, cantGas, cantGaso, cantC250
           ESCRIBA: "INGRESAR DATOS DE VEHÍCULOS:"
           ESCRIBA: "INGRESE LA CANTIDAD QUE INGRESA AL DÍA:"
           LEA: cant
           ESCRIBA: "INGRESE LA CANTIDAD DE VEHÍCULOS A GAS:"
```

```
        LEA: cantGas
        ESCRIBA: "INGRESE LA CANTIDAD DE VEHÍCULOS A GASOLINA:"
        LEA: cantGaso
        Vehiculo obj1=Vehiculo("TIPO VEHÍCULO", 5, cant, cantGas, cantGaso)

        ESCRIBA: "INGRESAR DATOS DE MOTOS:"
        ESCRIBA: "INGRESE LA CANTIDAD QUE INGRESA AL DÍA:"
        LEA: cant
        ESCRIBA: "INGRESE LA CANTIDAD DE MOTOS CON CILINDRAJE MAYOR A 250:"
        LEA: cantC250
        Moto obj2=Moto("TIPO MOTO", 2, cant, cantC250)

        obj1.mostrarVehiculo()
        obj2.mostrarMoto()
        ENTERO cantP=obj1.calcularPersonas( )+obj2.calcularPersonas( )
        ESCRIBA: "CANTIDAD MÁXIMA DE PERSONAS:", cantP
    FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Suponiendo que el usuario ingrese los siguientes datos, se crearán los objetos:

obj1: Vehículo
nombre= "TIPO VEHÍCULO" capacidadPasajeros=5 cantidad=1800 cantidadGas=200 cantidadGasolina=1600
MedioTransporte(CADENA nom, ENTERO cap, ENTERO cant) calcularPersonas(): ENTERO mostrarMedioTransporte() Vehiculo(CADENA nom, ENTERO cap, ENTERO cant, ENTERO cgas, ENTERO cgaso) mostrarVehiculo()

obj2: Moto
nombre= "TIPO MOTO" capacidadPasajeros=2 cantidad=500 cantCilindraje250=120
MedioTransporte(CADENA nom, ENTERO cap, ENTERO cant) calcularPersonas(): ENTERO mostrarMedioTransporte() Moto(CADENA nom, ENTERO cap, ENTERO cant, ENTERO cc250) mostrarMoto()

Salida

Los métodos de mostrar datos producen la siguiente salida:

```
NOMBRE: TIPO VEHÍCULO
CAPACIDAD DE PASAJEROS: 5
CANTIDAD  DE PASAJEROS QUE INGRESAN/DÍA: 1800
CANTIDAD  DE VEHÍCULOS A GAS: 200
CANTIDAD  DE VEHÍCULOS A GASOLINA: 1600

NOMBRE: TIPO MOTO
CAPACIDAD DE PASAJEROS: 2
CANTIDAD QUE INGRESAN/DÍA: 500
CANTIDAD DE MOTOS CON CILINDRAJE MAYOR A 250: 120

CANTIDAD MÁXIMA DE PERSONAS: 10000
```

7.14 Polimorfismo

El concepto de polimorfismo se aplica a varios objetos que comparten un método con el mismo nombre. Cuando este método es invocado depende del objeto que lo contiene; por lo tanto tendrá un comportamiento diferente para cada instancia.

Ejercicio resuelto Cap7-Ejer13

Se desea modelar la representación lógica de un carro mediante objetos circulares y rectangulares. En el caso más simple, un carro está compuesto por cuatro llantas y la carrocería. De cada una de las partes se debe almacenar el color y las medidas de los círculos y el rectángulo. Se debe mostrar por pantalla el área completa del carro.

1. **Identificación de datos, acciones y limitaciones**

- Datos de entrada: medidas y color de los círculos y del rectángulo.
- Datos de salida: área del carro.
- Acciones: capturar las medidas y el color de las figuras, calcular el área.
- Limitaciones: El carro está representado solo por cuatro llantas y la carrocería.

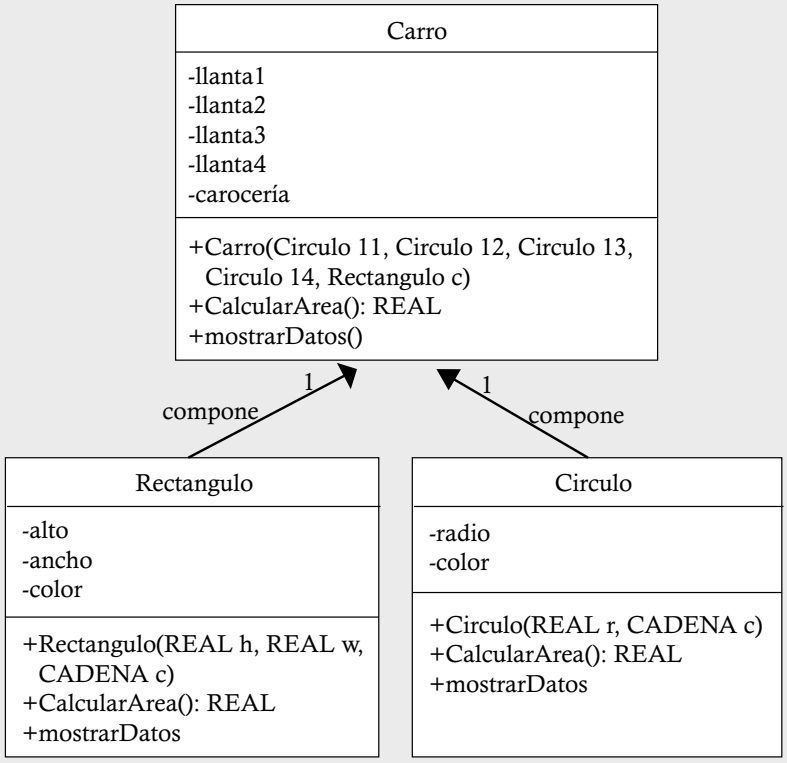
Continúa en la siguiente página

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): carro, círculo, rectángulo.
- Relación de los sustantivos con los datos y las acciones:
  Carro: descripción del carro formado con figuras, cálculo del área del carro.
  Círculo: medidas, color y cálculo del área del círculo.
  Rectángulo: medidas, color y cálculo del área del rectángulo.
  Clases seleccionadas: Carro, Círculo, Rectángulo
- Definición de constructores y destructores para las clases:
  Las tres clases tienen un constructor que recibe los valores por asignar a las variables de la clase.
- Diagramación de la clase con sus variables y métodos: La clase Carro tiene una relación de composición con las clases Circulo y Rectangulo, ya que las llantas y la carrocería componen el carro. El símbolo de composición es un rombo de color negro sobre el extremo de la clase compuesta. La relación se lee de la siguiente forma:
  - Un carro se compone de un rectángulo y de cuatro círculos.
  - Un rectángulo compone a un carro.
  - Un círculo compone a un carro.
  - Un carro se compone de un rectángulo y de cuatro círculos.
  - Un rectángulo compone a un carro.
  - Un círculo compone a un carro.

En el diagrama se puede observar el polimorfismo de los métodos mostrarDatos() y calcularArea(). Estos dos métodos se encuentran en todas las clases, pero tienen un comportamiento diferente en cada una.

Continúa en la siguiente página



- **Explicación de los métodos de la clase Carro:**
  Carro: Constructor de la clase.
  calcularArea: Método que suma las áreas de los componentes del carro.
  mostrarDatos: Método que muestra por pantalla las variables de la clase.
- **Explicación de los métodos de la clase Rectangulo:**
  Rectangulo: Constructor de la clase.
  calcularArea: Método que calcula el área de un rectángulo.
  mostrarDatos: Método que muestra por pantalla las variables de la clase.
- **Explicación de los métodos de la clase Circulo:**
  Circulo: Constructor de la clase.
  calcularArea: Método que calcula el área de un circulo.
  mostrarDatos: Método que muestra por pantalla las variables de la clase.

Continúa en la siguiente página

3. Definición del método principal
El método principal se encuentra en la clase Carro, donde se instancia un objeto de esta clase, se invoca el constructor y se calcula el área del carro.

Definición de variables:
llanta1, llanta2, llanta3, llanta4: Variables que almacenan objetos de tipo Circulo que representan las llantas del carro.
carrocería: Variable que almacena un objeto de tipo Rectangulo que representa la carrocería del carro.
cacharro: Variable que almacena un objeto de tipo carro.
i: Variable que controla la ejecución del ciclo PARA, mediante el cual se ingresan los datos de las cuatro llantas.
r: Variable para almacenar el radio de las llantas.
al, an: Variables para almacenar el alto y ancho del objeto de tipo Rectangulo.
c: Variable para almacenar el color de las partes del carro.
area: Variable que almacena el área total del carro.

Algoritmo

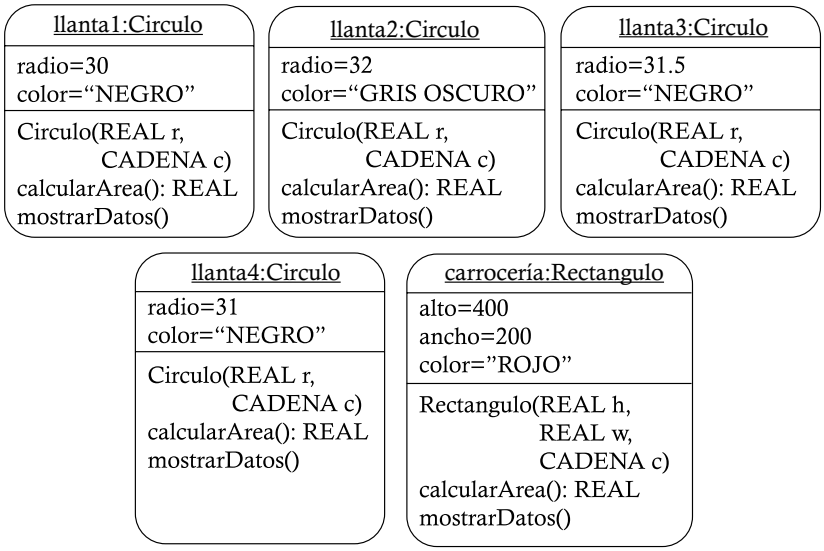
```
CLASE Rectangulo
    PRIVADO REAL alto
    PRIVADO REAL ancho
    PRIVADO CADENA color
    PUBLICO Rectangulo(REAL h, REAL w, CADENA c)
        alto=h
        ancho=w
        color=c
    FIN_Rectangulo
    PUBLICO REAL calcularArea()
         REAL area=alto*ancho
         RETORNE area
    FIN_calcularArea
    PUBLICO VOID mostrarDatos()
           ESCRIBA: “ALTO:”,  alto
           ESCRIBA: “ANCHO:”,  ancho
           ESCRIBA: “COLOR:”,  color
      FIN_mostrarDatos
FIN_CLASE
```

```
CLASE Circulo
      PRIVADO REAL radio
      PRIVADO CADENA color
      PUBLICO Circulo(REAL r, CADENA c)
          radio=r
          color=c
      FIN_Circulo

PUBLICO REAL calcularArea()
                REAL area=3.1416*radio^2
                RETORNE area
FIN_calcularArea
PUBLICO VOID mostrarDatos()
          ESCRIBA: "RADIO:", radio
          ESCRIBA: "COLOR:", color
      FIN_mostrarDatos
FIN_CLASE

CLASE Carro
    PRIVADO Circulo llanta1
    PRIVADO Circulo llanta2
    PRIVADO Circulo llanta3
    PRIVADO Circulo llanta4
    PRIVADO Rectangulo carroceria
    PUBLICO Carro(Circulo l1, Circulo l2, Circulo l3, Circulo l4, Rectangulo c)
            llanta1=l1
            llanta2=l2
            llanta3=l3
            llanta4=l4
            carroceria=c
      FIN_Carro
      PUBLICO REAL calcularArea()
            REAL area
            area=llanta1.calcularArea()+llanta2.calcularArea()+llanta3
            calcularArea()
                    + llanta4.calcularArea()+carrocería.calcularArea()
            RETORNE area
      FIN_calcularArea
      VOID mostrarDatos()
          ESCRIBA: "LLANTA 1"
          llanta1.mostrarDatos()
          ESCRIBA: "LLANTA 2"
          llanta2.mostrarDatos()
          ESCRIBA: "LLANTA 3"
```

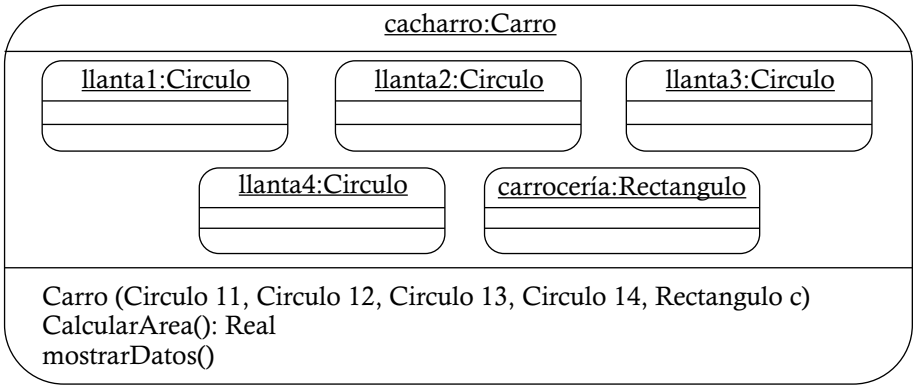
```
            llanta3.mostrarDatos()
            ESCRIBA: "LLANTA 4"
            llanta4.mostrarDatos()
            ESCRIBA: "CARROCERÍA"
            carroceria.mostrarDatos()
        FIN_mostrarDatos

        METODO PRINCIPAL()
                ENTERO i
                REAL r, al, an
                CADENA c
                Circulo llanta1, llanta2, llanta3, llanta4
                Rectangulo carroceria
                ESCRIBA: "CREANDO UN CARRO"
                PARA (i=1; i<=4;i=i+1) HAGA
                    ESCRIBA: "INGRESE LOS DATOS DE  LA LLANTA:", i
                    ESCRIBA: "INGRESE EL RADIO:"
                    LEA: r
                        ESCRIBA: "INGRESE EL COLOR:"
                        LEA: c
                        CASOS DE (i)
                                CASO 1: llanta1=Circulo(r,c)
                                CASO 2: llanta2=Circulo(r,c)
                                CASO 3: llanta3=Circulo(r,c)
                                CASO 4: llanta4=Circulo(r,c)
                        FIN_CASOS
                FIN_PARA
                ESCRIBA: "INGRESE LOS DATOS DE  LA CARROCERÍA"
                ESCRIBA: "INGRESE EL ALTO:"
                LEA: al
                ESCRIBA: "INGRESE EL ANCHO:"
                LEA: an
                ESCRIBA: "INGRESE EL COLOR:"
                LEA: c
                carroceria=Rectangulo(al,an,c)
                Carro cacharro=Carro(llanta1, llanta2, llanta3, llanta4, carroceria)
                cacharro.mostrarDatos ()
                REAL area=cacharro.calcularArea()
                ESCRIBA: "ÁREA DEL CARRO:", area
        FIN_PRINCIPAL
FIN_CLASE
```

Prueba de escritorio

Suponiendo como entrada las cuatro llantas y la carrocería:



La clase Carro, compuesta por los anteriores objetos, queda de la siguiente forma:



Salida

```
LLANTA 1
RADIO: 30
COLOR: NEGRO

LLANTA 2
RADIO: 32
COLOR: GRIS OSCURO

LLANTA 3
RADIO: 31.5
COLOR: NEGRO

LLANTA 4
RADIO: 31
COLOR: NEGRO

CARROCERÍA
ALTO: 400
ANCHO: 200
COLOR: ROJO

ÁREA DEL CARRO: 92180.7686
```

7.15 Otros modificadores de acceso

Existen otros modificadores de acceso que pueden ser aplicados a variables, métodos y clases. En la siguiente tabla se resumen los modificadores: FINAL, ESTATICO y ABSTRACTO.

	Variable	Método	Clase
FINAL	Variable constante o invariante.	Método que no se puede sobrescribir en una herencia de clases.	Clase de la cual no se puede heredar.
ESTATICO	Variable que se puede acceder sin tener una instancia de la clase. El contenido de la variable es compartido por todos los objetos instanciados a partir de la clase.	Método que se accede sin tener una instancia de la clase. Se invoca a partir del nombre de clase por medio del operador punto (.). Estos métodos no pueden acceder a variables de la clase que no sean estáticas también.	Solo aplica para lenguajes que lo soporten

Continúa en la siguiente página

	Variable	Método	Clase
ABSTRACTO	No aplica.	Métodos que no se implementan; solamente se definen. Estos métodos implementan su contenido en una clase heredada de la clase abstracta que los contiene.	Clase de la cual no se pueden crear objetos. Si una clase tiene un método abstracto, entonces la clase tiene que ser declarada abstracta también.

7.16 Clases abstractas

Son clases que no implementan todos los métodos que las conforman. Una clase se considera abstracta si tiene por lo menos un método sin implementar. Los métodos no implementados se deben declarar como abstractos y serán implementados en las subclases que lo utilicen. Si una clase tiene todos sus métodos abstractos se conoce como *interfaz*.

Ejercicio resuelto Cap7-Ejer14

Elaborar un programa que permita manipular cuentas bancarias de ahorro o crédito. De ambos tipos de cuentas se debe almacenar el nombre del cliente, la cédula y la fecha de apertura. De las cuentas de ahorro se debe almacenar el saldo actual y de las cuentas de crédito se debe almacenar el valor inicial del crédito, el valor de las cuotas y la cantidad de cuotas canceladas. Se debe permitir consultar el saldo de la cuenta y realizar las operaciones necesarias según el tipo de cuenta.

1. Identificación de datos, acciones y limitaciones

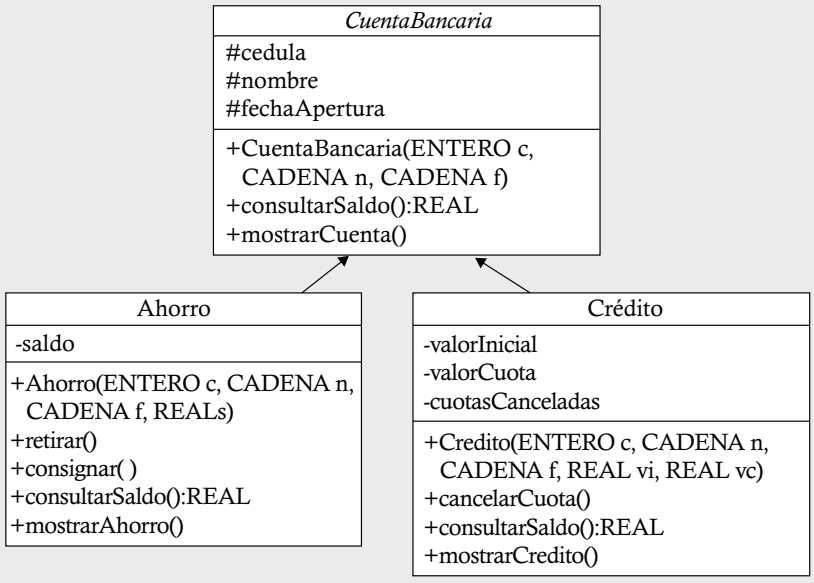
- Datos de entrada: nombre del cliente, la cédula, la fecha de apertura, saldo de cuenta ahorro, valor inicial del crédito, valor de las cuotas y cantidad de cuotas canceladas.
- Datos de salida: saldo.
- Acciones: capturar datos, mostrar datos, calcular saldo.
- Limitaciones: No se tienen en cuenta intereses.

2. Definición de clases

- Identificación de posibles clases (sustantivos relevantes): cuenta bancaria, ahorro, crédito, operaciones.

Continúa en la siguiente página

- Relación de los sustantivos con los datos y las acciones:
  Cuenta bancaria: nombre del cliente, la cédula, la fecha de apertura, consultar saldo.
  Ahorro: saldo, retirar, consignar.
  Crédito: valor inicial del crédito, valor de las cuotas, cantidad de cuotas canceladas, cancelar cuota.
  Operaciones: N/A.
  Clases seleccionadas: CuentaBancaria, Ahorro y Credito.
- Definición de constructores y destructores para las clases:
- Las tres clases tienen un constructor que recibe los parámetros de la clase.
- Diagramación de la clase con sus variables y métodos: Las clases Ahorro y Credito heredan de la clase CuentaBancaria. Como el cálculo del saldo depende del tipo de cuenta, este método es declarado como abstracto en la clase CuentaBancaria, de tal manera que las clases Ahorro y Crédito deben codificar este método. En el diagrama, la clase CuentaBancaria aparece en cursiva para indicar que es abstracta, al igual que el método calcularSaldo(). Adicionalmente, tenemos polimorfismo del método abstracto ya que tiene diferente comportamiento en las clases Ahorro y Credito, aunque en su declaración es igual.



Continúa en la siguiente página

- **Explicación de los métodos de la clase CuentaBancaria**:
  CuentaBancaria: Constructor que recibe el valor de las variables de la clase.
  consultarSaldo: Método abstracto para calcular el saldo de la cuenta. Tiene diferente comportamiento para una cuenta de ahorros y para una cuenta de crédito.
  mostrarCuenta: Método que muestra por pantalla las variables de la clase.

- **Explicación de los métodos de la clase Ahorro:**
  Ahorro: Constructor que recibe el valor de las variables de la clase.
  retirar: Método que le permite al usuario retirar dinero de su cuenta de ahorros siempre y cuando tenga saldo disponible. El saldo final no es un valor de retorno porque es una variable de la clase que se puede acceder en cualquier momento, siempre que se tenga permiso de acceso.
  consignar: Método que le permite al usuario consignar dinero en su cuenta de ahorros. El saldo final no es un valor de retorno porque es una variable de la clase.
  consultarSaldo: Método que retorna el saldo de la cuenta de ahorros.
  mostrarAhorro: Método que muestra por pantalla las variables de la clase.

- **Explicación de los métodos de la clase Credito:**
  Credito: Constructor que recibe el valor de las variables de la clase.
  cancelarCuota: Método que le permite al usuario cancelar una cuota de su crédito siempre y cuando tenga saldo pendiente. La cantidad de cuotas canceladas no es un valor de retorno porque es una variable de la clase.
  consultarSaldo: Método que retorna el saldo pendiente del crédito.
  mostrarCredito: Método que muestra por pantalla las variables de la clase.

3. Definición del método principal

El método principal se encuentra en una clase llamada Principal, donde el usuario debe seleccionar si desea manipular cuenta de ahorro o crédito. Para cada una de las cuentas se ofrecen diferentes opciones en un menú.

Continúa en la siguiente página

Definición de variables:
ced, nom y fe: Variables para almacenar los datos generales de la cuenta (cédula, nombre y fecha de apertura).
opcion: Variable que almacena la decisión del usuario de ingresar una cuenta de ahorros o crédito.
obj1: Variable que almacena un objeto de tipo Ahorro.
obj2: Variable que almacena un objeto de tipo Credito.
sal: Variable para almacenar el saldo inicial de una cuenta de ahorros.
vi y vc: Variables para almacenar el valor inicial y el valor de las cuotas de un crédito.

Algoritmo

```
ABSTRACTA CLASE CuentaBancaria
    PROTEGIDO ENTERO cedula
    PROTEGIDO CADENA nombre
    PRIVADO CADENA fechaApertura
    PÚBLICO CuentaBancaria (ENTERO c, CADENA n, CADENA f)
        cedula=c
        nombre=n
        fechaApertura=f
    FIN_CuentaBancaria

    PUBLICO ABSTRACTO REAL consultarSaldo()

    PUBLICO VOID  mostrarCuenta()
        ESCRIBA: "CÉDULA:",  cedula
        ESCRIBA: "NOMBRE:", nombre
        ESCRIBA: "FECHA DE APERTURA:", fechaApertura
    FIN_mostrarCuenta
FIN_CLASE

CLASE Ahorro HEREDA CLASE CuentaBancaria
    PRIVADO REAL saldo
    PÚBLICO Ahorro(ENTERO c, CADENA n, CADENA f, REAL s)
        CuentaBancaria(c, n, f)
        saldo=s
    FIN_Ahorro
    PUBLICO VOID retirar()
        REAL   valor
        ESCRIBA: "INGRESE EL VALOR POR RETIRAR:"
        LEA: valor
        SI (valor<=saldo) ENTONCES
```

```
                saldo=saldo-valor
            SI_NO
                ESCRIBA: "FONDOS INSUFCIENTES"
            FIN_SI
    FIN_retirar
    PUBLICO VOID consignar()
            REAL   valor
            ESCRIBA: "INGRESE EL VALOR POR CONSIGNAR:"
            LEA: valor
            saldo=saldo+valor
    FIN_consignar
    PUBLICO REAL consultarSaldo()
            RETORNE saldo
    FIN_consultarSaldo
    PUBLICO VOID mostrarAhorro()
            mostrarCuenta()
            ESCRIBA: "SALDO:", saldo
    FIN_mostrarAhorro
FIN_CLASE

CLASE Credito HEREDA CLASE CuentaBancaria
    PRIVADO REAL valorInicial
    PRIVADO REAL valorCuotas
    PRIVADO ENTERO cuotasCanceladas

    PUBLICO Credito(ENTERO c, CADENA n, CADENA f, REAL vi,
    REAL vc)
            CuentaBancaria(c, n, f)
            valorInicial=vi
            valorCuotas=vc
            cuotasCanceladas=0
    FIN_Credito

    PUBLICO VOID cancelarCuota( )
        SI (consultarSaldo()<=0)
            ESCRIBA: "SU DEUDA ESTÁ CANCELADA"
        SI_NO
            cuotasCanceladas=cuotasCanceladas+1
    FIN_cancelarCuota

PUBLICO REAL consultarSaldo()
            REAL saldo
            saldo=valorInicial-(valorCuotas*cuotasCanceladas)
            RETORNE saldo
    FIN_consultarSaldo
```

```
        PUBLICO VOID mostrarCredito()
                mostrarCuenta()
                ESCRIBA: "VALOR INICIAL DEL CRÉDITO:", valorInicial
                ESCRIBA: "VALOR DE LAS CUOTAS:", valorCuotas
                ESCRIBA: "CUOTAS CANCELADAS:", cuotasCanceladas
        FIN_mostrarCredito
FIN_CLASE

CLASE PRINCIPAL
        METODO PRINCIPAL()
                ENTERO ced, opcion
                CADENA nom, fe
                REAL sal, vi, vc
                ESCRIBA: "INGRESE LA CÉDULA:"
                LEA: ced
                ESCRIBA: "INGRESE EL NOMBRE:"
                LEA: nom
                ESCRIBA: "INGRESE LA FECHA DE APERTURA:"
                LEA fe

                ESCRIBA: "DIGITE 1 PARA CUENTA DE AHORROS Y 2
                PARA CUENTA DE CRÉDITO"
                LEA: opcion
                SI (opcion==1) ENTONCES
                        ESCRIBA: "INGRESE EL SALDO INICIAL"
                        LEA: sal
                        Ahorro obj1=Ahorro(ced, nom, fe, sal)
                        HAGA
                                ESCRIBA: "1. RETIRAR"
                                ESCRIBA: "2. CONSIGNAR"
                                ESCRIBA: "3. CONSULTAR SALDO"
                                ESCRIBA: "4. MOSTRAR CUENTA"
                                ESCRIBA: "5. TERMINAR"
                                LEA: opcion
                                CASOS  DE opcion
                                        CASO 1: obj1.retirar()
                                        CASO 2: obj1.consignar()
                                        CASO 3: REAL s=obj1 consultarSaldo()
                                            ESCRIBA: "SALDO CUENTA
                                                    DE AHORROS", s
                                        CASO 4: obj1.mostrarAhorro()
                                FIN_CASOS
                        MIENTRAS opcion<>5
                 SI_NO
                        ESCRIBA: "INGRESE EL VALOR DEL CRÉDITO"
```

```
                    LEA: vi
                    ESCRIBA: "INGRESE EL VALOR DE LAS  CUOTAS"
                    LEA: vc
                    Credito obj2=Credito(ced, nom, fe, vi, vc)
                    HAGA
                            ESCRIBA: "1. CANCELAR CUOTA"
                            ESCRIBA: "2. CONSULTAR SALDO"
                            ESCRIBA: "3. MOSTRAR CUENTA"
                            ESCRIBA: "4. TERMINAR"
                            LEA: opcion
                            CASOS  DE opcion
                                 CASO 1: obj2.cancelarCuota()
                                 CASO 2: REAL s=obj2.consultarSaldo()
                                    ESCRIBA: "SALDO DELCRÉDITO:", s
                                 CASO 3: obj2.mostrarCredito()
                            FIN_CASOS
                    MIENTRAS opcion<>4
            FIN_SI
        FIN_PRINCIPAL
    FIN_CLASE
```

Prueba de escritorio

Suponiendo que el usuario seleccione una cuenta de crédito, se creará el siguiente objeto:

obj2: Credito
cedula=12345 nombre="CAMILA LÓPEZ" fechaApertura= "OCTUBRE 12 DE 2013" valorInicial=5.000.000 valorCuota=76.000 cuotasCanceladas=0
CuentaBancaria(ENTERO c, CADENA n, CADENA f) mostrarCuenta() Credito(ENTERO c, CADENA n, CADENA f, REAL vi, REAL vc) cancelarCuota( ) consultarSaldo(): REAL mostrarCredito()

Salida
Si el usuario ingresa la opción 2, la salida será:

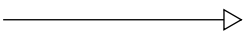
```
SALDO DEL CRÉDITO: 5000000
```

7.17 Aspectos para tener en cuenta

- Una clase puede no tener variables o no tener métodos.
- Por seguridad, se recomienda declarar las variables como privadas o protegidas.
- Para acceder las variables privadas, se recomienda implementar métodos accesores.
- Se recomienda leer valores por fuera de los métodos de la clase, de manera que puedan ser verificados en el método principal antes de ser enviados a la clase.
- Se recomienda no mostrar valores directamente dentro de los métodos de la clase, sino retornarlos para que sean mostrados por pantalla en el método principal. De esta manera se logra independencia de la interfaz gráfica.
- La sobrecarga de métodos se presenta en la misma clase.
- El polimorfismo se presenta sobre métodos en diferentes clases.
- Las clases heredadas pueden acceder solo los atributos y métodos visibles de la clase base, es decir, los públicos o protegidos.
- Si una clase tiene un método abstracto, la clase debe ser declarada como abstracta también.
- Se pueden presentar varias relaciones entre clases:
  - Asociación: Relaciona objetos que colaboran entre sí.

    1..* 1..*

  - Herencia: La subclase hereda los atributos y métodos visibles de la superclase.

  - Composición: Una clase está compuesta de variables que representan otras clases.

-Agregación: Una clase utiliza a otro objeto para su funcionamiento, mas sus existencias no son dependientes.

7.18 Ejercicios propuestos

1. Identifique los atributos y las acciones para cada uno de los siguientes enunciados y elabore el diagrama de cada clase:
    - Un trabajador de una empresa.
    - Un participante de una carrera atlética.
    - Un cliente en una funeraria.
2. Defina la clase Cuenta Bancaria que permita las operaciones: retirar, depositar, consultar saldo, entre otras. No olvide definir el constructor para poder crear objetos de dicha clase; además, recuerde que la cuenta siempre tiene un dueño.
3. Una entidad bancaria quiere disponer de una aplicación que cree una cuenta de un cliente y luego permita realizar movimientos sobre ella hasta que el usuario responda que no quiere realizar más. Cuando esto ocurra se debe mostrar en pantalla el nombre del usuario, su número de identificación, el saldo de la cuenta y la cantidad de movimientos de depósito y de retiro realizados por él.
4. La escuela de computación desea llevar a cabo un registro de ingreso de sus alumnos al laboratorio de programación haciendo uso de la fecha en la cual cada estudiante ingresa al lugar. Se requiere crear la clase Fecha con las siguientes especificaciones:
    - Variables: día, mes, año.
    - Constructores: uno que se inicialice con una fecha fija y otro que reciba como parámetros los valores para crear el objeto.
    - Métodos: modificar la fecha, mostrar fecha usando el formato: día/mes/año y mostrar la fecha en pantalla poniendo el mes en palabras.
5. Una entidad de promoción de la salud quiere hacer una campaña de prevención de enfermedades cardiovasculares sobre un grupo de personas con base en el índice de masa corporal. Elabore una aplicación que permita calcular y mostrar la clasificación de cada persona de acuerdo con la tabla establecida por la Organización Mundial de la Salud (OMS), la cual se puede ver en la dirección: http://goo.gl/fk3akg.
6. El departamento escolar de la facultad de ingeniería necesita obtener un reporte global de los promedios de sus n alumnos y que muestre el pro-