OBJETIVOS DE LA MATERIA

- Desarrollar la capacidad de razonamiento y lógica necesarias para Programar.
- Abordar problemas reales, analizarlos, definir la estrategia de su resolución, explicitar los algoritmos necesarios y codificarlos en un lenguaje de programación.
- Lograr entrenamiento en:
    1. Interpretar el problema (ANALISIS)
    2. Reducirlo a lo esencial (ABSTRACCION)
    3. Formular la estrategia de resolución (DIAGRAMA)
    4. Verificar dicha estrategia (PRUEBA DE ESCRITORIO)
    5. Codificar en un lenguaje de programación.
    6. Depurar errores de codificación.
    7. Pruebas, correcciones,...

1. COMPRENSION DEL PROBLEMA (ANALISIS)

El análisis del problema va a depender de la visión y del método que se utilice para interpretarlo. Estos métodos o modelos básicos son los que se llaman paradigmas de programación y le indican al programador como debe encarar la visión del mundo real. Un paradigma de programación es un modelo básico de análisis, diseño e implementación de programas.

2. ABSTRACCION

Describir lo esencial de un proceso haciendo abstracción de los detalles, para poder asimilarlo a otro y hacer uso de técnicas y conocimientos ya adquiridos. Una vez que contamos con todos los elementos necesarios claramente definidos podemos proceder a desarrollar la **estrategia** que nos llevará a solucionar el problema.

3. ESTRATEGIA

Es un plan cuidadosamente elaborado que describe en líneas generales, el conjunto de pasos a seguir para llegar a los resultados deseados. Estos pasos constituirán módulos de programación independientes. Dependiendo de la orientación o paradigma que se esté utilizando diremos que estamos en presencia de **Funciones** (Programación estructurada) o **Métodos** de clases (Programación Orientada a Objetos). Tanto en uno como en otro caso es fundamental conocer que es lo que el lenguaje ya trae incorporado (Bibliotecas de funciones, paquetes de clases).

4. VERIFICAR LA ESTRATEGIA (PRUEBA DE ESCRITORIO)

5. CODIFICAR EN UN LENGUAJE DE PROGRAMACION.

6. DEPURAR ERRORES DE CODIFICACION.

7. PRUEBAS, CORRECCIONES,...

1. ANALISIS DEL PROBLEMA

El propósito del análisis de un problema es posibilitar su posterior formulación en un lenguaje de programación.

- ✓ En primer lugar es esencial que las especificaciones de entrada (Datos) y salida (Resultados) sean descritas con detalle. Normalmente se parte de las especificaciones de salida, o sea de los resultados que se requieren.

- ✓ En segundo lugar necesitamos de un conocimiento detallado de las vinculaciones existentes entre datos y resultados.

  Dependiendo de que lenguaje de programación usemos tenemos diferentes caminos:

- ✓ Si usamos un lenguaje de la llamada Vertiente Declarativa, debemos describir los datos, sus vinculaciones y formular las metas (Resultados) pretendidos. El proceso de resolución es responsabilidad del lenguaje de programación.

- ✓ Si usamos un lenguaje de la Vertiente Imperativa u Orientación Objetos, debemos describir los datos , los resultados y todos el procedimiento para obtener estos últimos. El proceso de resolución estará constituido por uno o varios programas. Cada programa estará a su vez conformado por un principal (main()) que activara a funciones (Programación estructurada) o emitirá mensajes a métodos de clases (Orientación

Objetos). En este caso, el proceso de resolución es nuestra responsabilidad. En nuestra asignatura codificaremos este proceso en lenguaje Java, Orientación Objetos. En los ejemplos que sea conveniente, incluiremos también resoluciones en modalidad estructurada.

Vamos a un ejemplo:

Necesitamos conocer la superficie de un círculo y la longitud de su circunferencia.

1. COMPRENSION DEL PROBLEMA (ANALISIS)

En este caso tan sencillo es evidente que el dato de entrada será el radio y los resultados la superficie y el perímetro.

2. ABSTRACCION

Para resolver este problema **utilizando la filosofía de objetos** podríamos seguir los siguientes pasos:

- ✓ **Identificar el objeto:** identificar la entidad que engloba al problema, en nuestro ejemplo es el **círculo.**

- ✓ **Identificar sus atributos:** podemos decir que todo círculo tiene como datos esenciales el radio, (dato) además, nuestro problema en particular obtendrá el valor de la superficie y de la longitud, (resultados) que **pueden figurar como atributos o no** dependiendo si se quiere guardar su estado en el objeto. Como nuestro enunciado no nos restringe la forma de implementarlo vamos a definir como atributos el radio, la superficie y la longitud.

3. ESTRATEGIA

- ✓ **Identificar sus métodos:** los métodos a elegir siempre dependerán de las operaciones que debemos realizar con los datos para obtener los resultados y exhibirlos, entonces podemos enumerar:

Inicializar el radio: Inicializar en 0, por ejemplo, el radio.
Inicializar la superficie: Inicializar en 0, por ejemplo, la superficie.
Inicializar la longitud: Inicializar en 0, por ejemplo, la longitud.
Ingresar el radio: Se ingresa solo el valor del radio ya que el resto de los atributos se calculan a partir de este.
Mostrar el radio: Mostrar el valor del radio.
Mostrar la superficie: Mostrar el valor de la superficie.
Mostrar la longitud: Mostrar el valor de la longitud.
Calcular la superficie: formula que devuelve la superficie
Calcular la longitud: formula que devuelve la longitud.

INTRODUCCIÓN AL ANALISIS Y PROGRAMACION ORIENTADA A OBJETOS

La orientación a objetos es una forma natural de pensar en relación con el mundo y de escribir programas de computación. Mire a su alrededor. Por todas partes: *¡objetos!* Personas, animales, plantas, automóviles, aviones, edificios, cortadoras de pastos, computadoras y demás. Cada una de ellas tiene ciertas características y se comporta de una manera determinada. Si las conocemos, es porque tenemos el ***concepto de lo que son****.* Conceptos persona, objetos persona. Los seres humanos **pensamos en términos de objetos**. Tenemos la capacidad maravillosa de la ***abstracción****,* que nos permite ver una imagen en pantalla como personas, aviones, árboles y montañas, en vez de puntos individuales de color.

Todos estos objetos tienen algunas cosas en común. **Todos tienen atributos***,* como tamaño, forma, color, peso y demás. Todos ellos exhiben **algún comportamiento.** Un automóvil acelera, frena, gira, etcétera. El objeto persona habla, ríe, estudia, baila, canta ...

Los seres humanos aprenden lo relacionado con los objetos estudiando sus atributos y observando su comportamiento. Objetos diferentes pueden tener muchos atributos iguales y mostrar comportamientos similares. Se pueden hacer comparaciones, por ejemplo, entre bebes y adultos, entre personas y chimpancés. Automóviles, camiones, pequeños carros rojos y patines tienen mucho en común.

La ***programación orientada a objetos*** *(POO)* **hace modelos de los objetos del mundo real** mediante sus contrapartes en software. Aprovecha las relaciones de clase, donde objetos de una cierta clase, como la clase de vehículos, tienen las mismas características. Aprovecha las relaciones de *herencia,* donde clases recién creadas de objetos se derivan heredando características de clases existentes, pero

también poseyendo características propias de ellos mismos. Los bebes tienen muchas características de sus padres, pero ocasionalmente padres de baja estatura tienen hijos altos.

La programación orientada a objetos nos proporciona una forma más **natural e intuitiva** de observar el proceso de programación, es decir *haciendo modelos* de objetos del mundo real, de sus atributos y de sus comportamientos. POO también hace modelos de la comunicación entre los objetos. De la misma forma que las personas se envían *mensajes* uno al otro los objetos también se comunican mediante mensajes.

La **POO *encapsula* datos (atributos) y funciones (comportamiento)** en paquetes llamados ***objetos****;* los datos y las funciones de un objeto están muy unidos. Los objetos tienen la propiedad de *ocultar* la *información.* Esto significa que aunque los objetos puedan saber cómo comunicarse unos con otros mediante *interfaces* bien definidas, a los objetos por lo regular no se les está permitido saber cómo funcionan otros objetos. Los detalles de puesta en práctica quedan ocultos dentro de los objetos mismos. (*Casi estamos diciendo que existe entre ellos el respeto a la intimidad …*) A esto se le llama ***Encapsulamiento***.

QUE ES LA PROGRAMACION ORIENTADA A OBJETOS

Es una técnica o estilo de programación que utiliza objetos como bloque esencial de construcción. Los programas se organizan como colecciones de **objetos** que colaboran entre sí enviándose mensajes. Solo se dispone de "***objetos que colaboran entre sí***". Por lo tanto un programa orientado a objetos viene definido por la ecuación:

Objetos + Mensajes = Programa

COMPONENTES BÁSICOS DE LA POO

Objetos

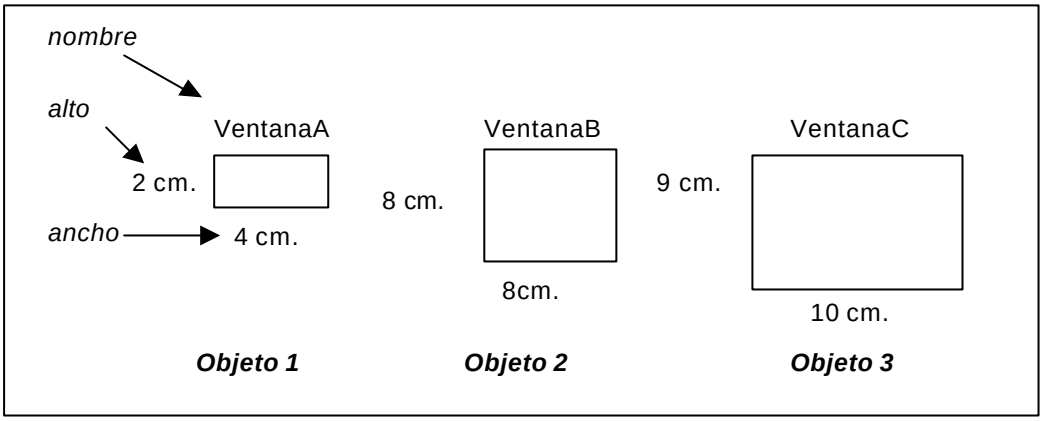
Es el elemento fundamental de la programación orientada a objetos. Es una entidad que posee atributos y métodos, los atributos definen el estado de mismo y los métodos definen su comportamiento.

Cada objeto forma parte de una organización, no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

¿Qué son objetos en POO? La respuesta es cualquier entidad del mundo real que se pueda imaginar:

- *Objetos físicos*
  - Automóviles en una simulación de tráfico.
  - Aviones en un sistema de control de tráfico aéreo.
  - Componentes electrónicos en un programa de diseño de circuitos.
  - Animales mamíferos.
- *Elementos de interfaces gráficos de usuarios*
  - Ventanas.
  - Objetos gráficos (líneas, rectángulos, círculos).
  - Menús.
- *Estructuras de datos*
  - Vectores.
  - Listas.
  - Arboles binarios.
- *Tipos de datos definidos por el usuario*
  - Números complejos.
  - Hora del día.
  - Puntos de un plano.

Como ejemplo de objeto, podemos decir que una ventana es un objeto que puede tener como atributos: **nombre, alto, ancho, etc.** y como métodos: **crear la ventana, abrir la ventana, cerrar la ventana, mover la ventana, etc.**



Métodos

Podemos definir un método como un programa procedimental escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Mensajes

Un mensaje es simplemente una petición de un objeto a otro para que éste se comporte de una determinada manera, ejecutando uno de sus métodos. La técnica de enviar mensajes se conoce como *paso de mensajes*.

Los mensajes relacionan unos objetos con otros y con el mundo exterior.

La figura 1. Representa un objeto A (emisor) que envía un mensaje Test al objeto B (receptor).

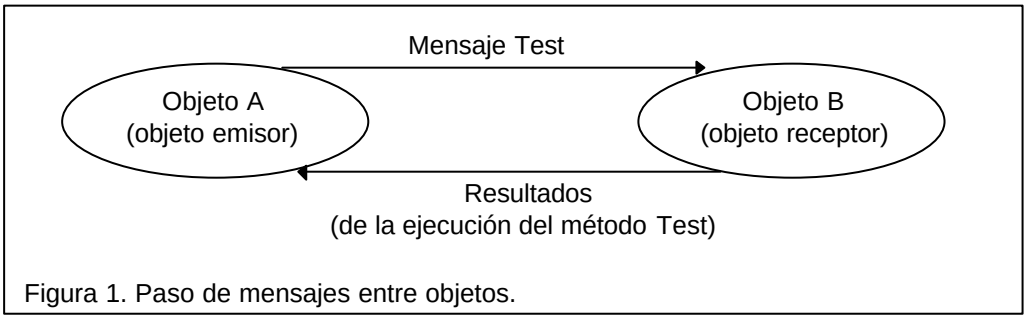


Figura 1. Paso de mensajes entre objetos.

Usando el ejemplo anterior de ventana, podríamos decir que algún objeto de Windows encargado de ejecutar aplicaciones, por ejemplo, envía un mensaje a nuestra ventana para que se abra. En este ejemplo el objeto emisor es un objeto de Windows y el objeto receptor es nuestra ventana, el mensaje es la petición de abrir la ventana y la respuesta es la ejecución del método abrir ventana.

Clases

Una clase es simplemente un modelo que se utiliza para describir uno o más objetos el mismo tipo.

Cada vez que se construye un objeto de una clase, se crea una instancia de esa clase. Por consiguiente, los objetos son instancias de clases. En general, los términos objetos e instancias de una clase se pueden utilizar indistintamente.

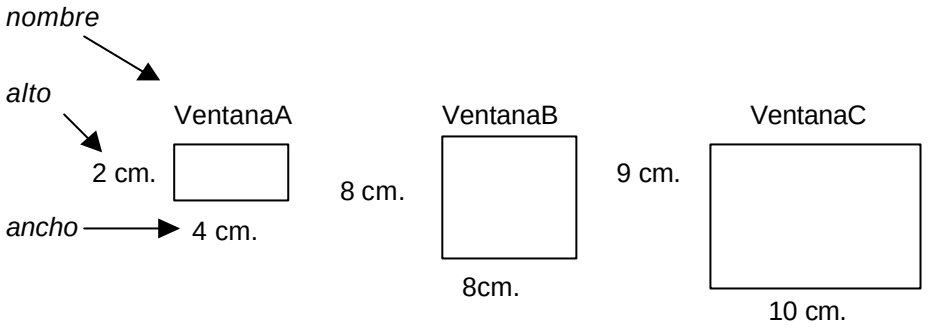
Una clase puede tener muchas instancias y cada una es un objeto independiente.

Siguiendo con el ejemplo de ventana, el modelo o clase para todas las ventanas sería:

Clase ventana

Atributos: nombre, alto, ancho.
Métodos: crear, abrir, cerrar, cambiar tamaño.

Objetos de tipo ventana



Objeto 1 ***Objeto 2*** ***Objeto 3***

CARACTERISTICAS DE LA POO

Abstracción

La abstracción se define como la "*extracción de las propiedades esenciales de un concepto*". Permite no preocuparse de los detalles no esenciales. Implica la identificación de los atributos y métodos de un objeto.

Es la capacidad para encapsular y aislar la información de diseño y ejecución.

Encapsulamiento

Toda la información relacionada con un objeto determinado está agrupada de alguna manera, pero el objeto en sí es como una caja negra cuya estructura interna permanece oculta, tanto para el usuario como para otros objetos diferentes, aunque formen parte de la misma jerarquía. La información contenida en el objeto será accesible sólo a través de la ejecución de los métodos adecuados.

Herencia

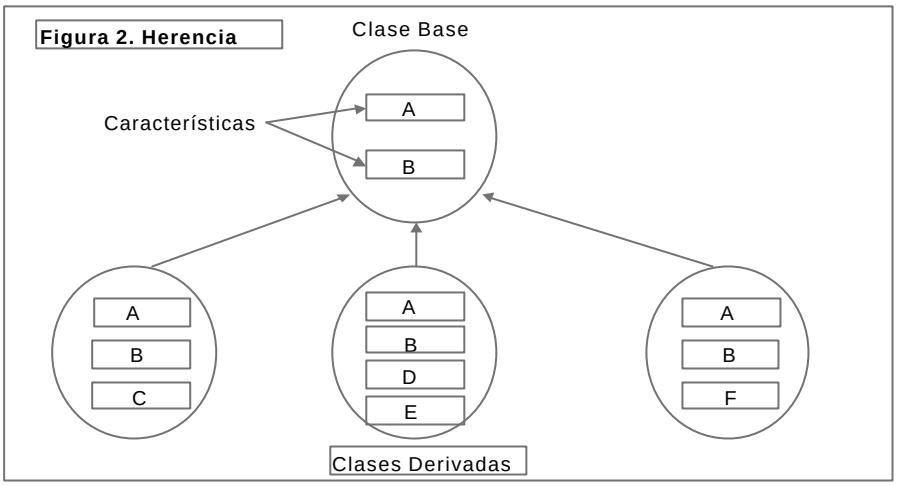
La herencia es la propiedad que permite a los objetos construirse a partir de otros objetos. El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases. Así por ejemplo, las clases de animales se dividen en mamíferos, anfibios, insectos, pájaros, etc. La clase de vehículos se divide en automóviles, autobuses, camiones, motocicletas, etc.

El principio de este tipo de división es que cada subclase comparte características comunes con la clase de la que se deriva. Los automóviles, camiones autobuses y motocicletas -que pertenecen a la clase vehículo- tienen ruedas y un motor; son las características de vehículos. Además de las características compartidas con otros miembros de la clase, cada subclase tiene sus propias características particulares: autobuses, por ejemplo, tienen un gran número de asientos, un aparato de televisión para los viajeros, mientras que las motocicletas tienen dos ruedas, un manillar y un asiento doble.

La idea de herencia se muestra en la figura 2. Observen en la figura que las características A y B que pertenecen a la clase base, también son comunes a todas las clases derivadas, y a su vez estas clases derivadas tienen sus propias características.

La herencia impone una relación jerárquica entre clases en la cual una clase *hija* hereda de su clase *padre*. Si una clase sólo puede recibir características de otra clase base, la herencia se denomina *herencia simple*.

Si una clase recibe propiedades de más de una clase base, la herencia se denomina *herencia múltiple*.



Como ejemplo, supongamos que tenemos que registrar los datos de alumnos y profesores, para ello vamos a analizar el problema y refinar la solución en varios pasos:

Primer paso: Identificamos los atributos esenciales para cada entidad.

Alumno	**Profesor**
Nombre	Nombre
Domicilio	Domicilio
Telefono	Telefono
Legajo	Legajo
Carrera	Titulos
Materias	Cursos

Segundo paso: Si observamos, los dos tienen atributos en común: Nombre, Domicilio y Telefono que se corresponden con los datos personales de cualquier persona, y datos particulares para el alumno y el docente. Entonces podríamos escribir los atributos de la siguiente forma:

Alumno	**Profesor**
Datos personales	Datos personales
Datos del alumno	Datos del profesor

Tercer paso: Si agrupamos los datos personales en una clase llamada Persona con los datos comunes a las dos entidades, formaremos lo siguiente:

Persona

Datos Personales

Alumno	**Profesor**
Es una persona	Es una persona
(porque tiene datos de persona)	(porque tiene datos de persona)
Tiene datos del alumno	Tiene datos del profesor

Cuarto paso: Si completamos los atributos, vamos a obervar que hemos ahorrado atributos, ya que en el primer paso había atributos que se repetían en las dos entidades y ahora hay tres entidades en donde cada una tiene los atributos que le corresponden sin repetición, pero alumno y profesor van a heredar de persona los atributos que le correspondan.

Persona

Nombre
Domicilio
Telefono

Alumno	**Profesor**
Hereda los datos de persona	Hereda los datos de persona
Legajo	Legajo
Carrera	Titulos
Materias	Cursos

Polimorfismo

En un sentido literal, *polimorfismo* significa cualidad de tener más de una forma. En el contexto de POO, el polimorfismo se refiere al hecho que una misma operación puede tener diferente comportamiento en diferentes objetos. En otras palabras, diferentes objetos reaccionan al mismo mensaje de modo diferente.

Por ejemplo, consideremos la operación sumar. En un lenguaje de programación el operador + representa la suma de dos números (x + y) de diferentes tipos: enteros, coma flotante, ... Además se puede definir la operación de sumar dos cadenas : concatenación mediante el operador suma.

De modo similar, supongamos un número de figuras geométricas que responden todas al mensaje, **dibujar**. Cada objeto reacciona a este mensaje visualizando su figura en la pantalla de visualización. Obviamente, el mecanismo real para visualizar los objetos difiere de una figura a otra, pero todas las figuras realizan esta tarea en respuesta al mismo mensaje.

Otro ejemplo podría ser el siguiente: un usuario de un sistema tiene que imprimir un listado de sus clientes, el puede elegir imprimirlo en el monitor, en la impresora, en un archivo en disco o en una terminal remota. De esta forma el mensaje es imprimir el listado pero la respuesta la dará el objeto que el usuario desee, el monitor, la impresora, el archivo o la terminal remota. Cada uno responderá con métodos (comportamientos) diferentes.

Resumen

Como vimos anteriormente, el análisis de un problema es el paso más importante para resolver un problema, existen varias formas de plantear las soluciones, pero hay dos grandes formas que se están utilizando hoy en día, que son el análisis estructurado y el orientado a objetos, el primero se está dejando de usar, sin embargo aún quedan muchos sistemas resueltos bajo esta óptica que hay que mantener o actualizar. Y la segunda forma es la que nosotros vamos a usar ya que es la tendencia del mercado, es la que mejor rendimiento tiene, se está usando desde hace varios años y es la que tiene mayor soporte en cuanto a tecnologías de análisis, diseño y desarrollo de software.

A continuación vamos a realizar una breve comparación entre la programación estructurada y la orientada a objetos que nos servirá para entender mejor como tendremos que resolver los problemas bajo la óptica orientada a objetos.

Programación estructurada

La programación estructurada tiende a ser ***orientada a* la *acción.*** Los programadores se **concentran en escribir funciones**, que son grupos de acciones que ejecutan alguna tarea común y que se agrupan para formar programas. La **unidad fundamental de la programación es la función**. Es cierto que los datos son importantes, pero la óptica es que los datos son materia prima para las acciones que las funciones ejecutan. Semánticamente las acciones son verbos. Los ***verbos*** en una especificación de sistema ayudan al programador a determinar el conjunto de funciones que juntas funcionarán para ponerlo en marcha.

Programación orientada a objetos

La programación orientada a objetos como su nombre lo indica está ***orientada al objeto (concepto).*** El **concepto** se implementa en clases (*class*). **La clase es la unidad básica de programación.** Es una unidad que contiene los atributos y todas las funciones necesarias a su tratamiento.

Entonces cada clase contiene datos junto con un conjunto de funciones que manipula dichos datos. Los componentes de datos de una clase se llaman ***datos miembros*** o *atributos.* El comportamiento de la clase se implementa mediante ***funciones miembro***.

El foco de atención está sobre los objetos, en vez de sobre las funciones. Los ***sustantivos*** en una especificación de sistema ayudan al programador a determinar el conjunto de clases, a partir de las cuales serán creados los objetos que funcionarán conjuntamente para poner en práctica el sistema.

Es muy importante que a la hora de analizar un problema utilicemos sustantivos y no verbos.

A continuación resolveremos un problema bajo la óptica orientada a objetos:

PROBLEMA

Dado el valor de los tres lados de un triángulo, calcular el perímetro.

COMPRENSION DEL PROBLEMA (ANALISIS)

En este caso tan sencillo es evidente que el dato de entrada serán los lados y el resultado el perímetro.

ABSTRACCION

Identificar el objeto: identificar la entidad que engloba al problema, en nuestro ejemplo es el triángulo.

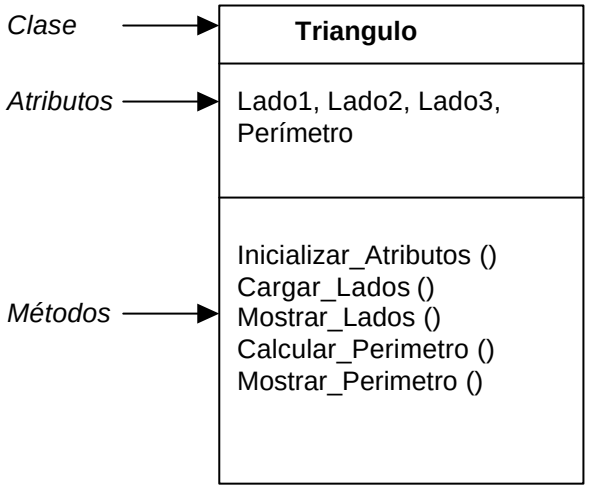
Identificar sus atributos: podemos decir que todo triángulo tiene como datos esenciales los tres lados, además, nuestro problema en particular necesitará el valor del perímetro, que es un datos calculable a partir de los tres lados y que puede figurar como atributo o no dependiendo si se quiere guardar su estado en el objeto. Como nuestro enunciado no nos restringe la forma de implementarlo vamos a definir como atributos los tres lados y el perímetro.

ESTRATEGIA

Identificar sus métodos: los métodos a elegir siempre dependerán de las operaciones que debemos realizar con los datos para obtener los resultados y exhibirlos, entonces podemos enumerar:

Inicializar los atributos: Inicializar en 0, por ejemplo, los lados y el perímetro.
Ingresar los lados: Se ingresa el valor de cada lado.
Mostrar los lados: Mostrar el valor de los lados.
Calcular el perímetro: formula que devuelve el perímetro del triángulo.
Mostrar el perímetro: Mostrar el valor del perímetro.

Graficando el resultado

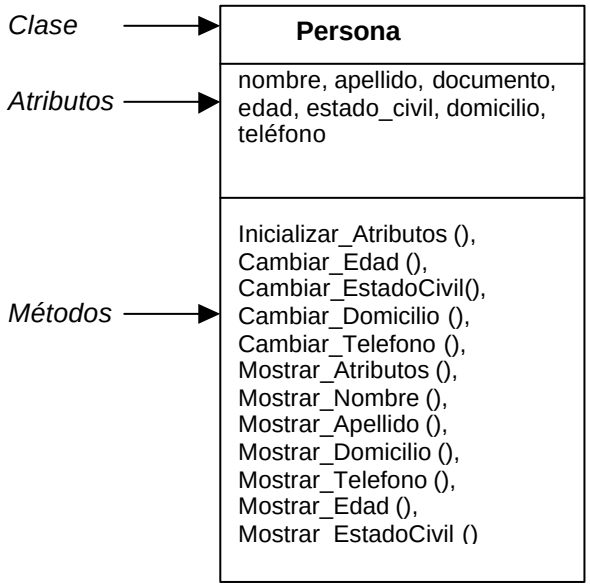


1. Dados las siguientes entidades definirlas, identificar la clase, los atributos, los métodos y los mensajes correspondientes.

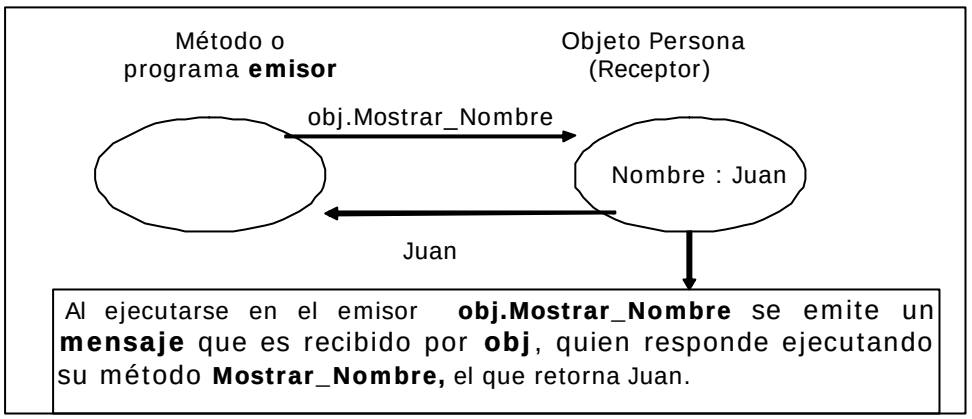
1. Persona 2. Cliente 3. Proveedor 4. Producto
5. Computadora 6. Mesa 7. Automóvil 8. Avión
9. Cuenta bancaria 10. Comprobante de factura
11. Definir cualquier entidad de la vida real, que se utilice en la facultad, trabajo, hogar, etc.

Resoluciones

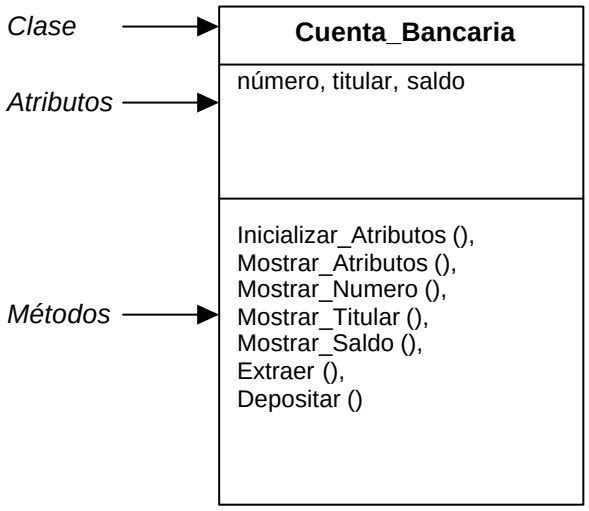
1.1. Persona



Ej : 1 Procesamiento de un mensaje.



1.9. Cuenta bancaria



LENGUAJE JAVA

Java es un lenguaje orientado a objetos creado por James Gosling e introducido por Sun Mycrosystems en junio de 1995. Fue diseñado como un lenguaje de sintaxis muy similar a la del C++, pero con ciertas características diferentes que lo hacen más simple y portable a través de diferentes plataformas y sistemas operativos (tanto a nivel de código fuente como nivel binario), para lo cual se implementó como un lenguaje híbrido, a medio camino entre compilado e interpretado. A grandes rasgos, se compila el código fuente *java a un formato binario *.class *(bytecode),* que luego es interpretado por una máquina virtual java, la cual debe estar implementada para la plataforma particular usada.

En Java podemos construir dos tipos básicos de programas: utilizarlo como un lenguaje de propósito general para construir aplicaciones independientes o emplearlo para crear *applets* (tema que se verá en la asignatura Paradigmas).

CARACTERISTICAS

Hay muchas razones por las que Java es tan popular y útil. Aquí se resumen algunas características importantes:

- **Orientación a objetos**: Java está totalmente orientado a objetos. No hay funciones sueltas en un programa de Java. Todos los métodos se encuentran dentro de clases. Los tipos de datos primitivos, como los enteros o dobles, tienen empaquetadores de clases, siendo estos objetos por sí mismos, lo que permite que el programa las manipule.
- **Simplicidad**: la sintaxis de Java es similar a ANSI C y C++ y, por tanto, fácil de aprender; aunque es mucho más simple y pequeño que C++. Elimina encabezados de archivos, preprocesador, aritmética de apuntadores, herencia múltiple, sobrecarga de operadores, struct, union y plantillas. Además, realiza automáticamente la recolección *de* basura, lo que hace innecesario el manejo explícito de memoria.

- **Robustez**: Por la forma en que está diseñado, Java no permite el manejo directo del hardware ni de la memoria (no permite modificar valores de punteros, por ejemplo), de modo que se puede decir que es virtualmente imposible "colgar" un programa Java. El intérprete siempre tiene el control. De hecho, el compilador es suficientemente inteligente como para no permitir una serie de acciones que podrían traer problemas, como usar variables sin inicializarlas, modificar valores de punteros directamente, acceder a métodos o variables en forma incorrecta, utilizar herencia múltiple, etc.
- **Compactibilidad**: Java está diseñado para ser pequeño. La versión más compacta puede utilizarse para controlar pequeñas aplicaciones. El intérprete de Java y el soporte básico de clases se mantienen pequeños al empaquetar por separado otras bibliotecas.
- **Portabilídad**: sus programas se compilan en el código de bytes de arquitectura neutra y se ejecutarán en cualquier plataforma con un intérprete de Java. Su compilador y otras herramientas están escritas en Java. Su intérprete está escrito en ANSI C. De cualquier modo, la especificación del lenguaje Java no *tiene* características *dependientes de* la implantación.
- **Amigable para el trabajo en red**: Java tiene elementos integrados para comunicación en red, applets Web, aplicaciones cliente-servidor, además de acceso remoto a bases de datos, métodos y programas.
- **Soporte a GUI**: la caja de herramientas para la creación de ventanas abstractas -(Abstract Windowing Toolkit) de Java simplifica y facilita la escritura de programas GUI (Graphic User Interface) orientados a eventos con muchos componentes de ventana.
- **Carga y vinculación incremental dinámica**: las clases de Java se vinculan dinámicamente al momento de la carga. Por tanto, la adición de nuevos métodos y campos de datos a clases, no requieren de recompilación de clases del cliente.
- **Internacionalización:** los programas de Java están escritos en Unicode, un código de caracter de 16 bits que incluye alfabetos de los lenguajes más utilizados en el mundo. La manipulación de los caracteres de Unicode y el soporte para fecha/hora local, etcétera, hacen que Java sea bienvenido en todo el mundo.
- **Hilos**: Java proporciona múltiples flujos de control que se ejecutan de manera concurrente dentro de uno de sus programas. Los hilos permiten que su programa emprenda varias tareas de cómputo al mismo tiempo, una característica que da soporte a programas orientados a eventos, para trabajo en red y de animación.
- **Seguridad**: entre las medidas de seguridad de Java se incluyen restricciones en sus applets, implantación redefinible de sockets y objetos de administrador de seguridad definidos por el usuario. Hacen que las applets sean confiables y permiten que las aplicaciones implanten y se apeguen a reglas de seguridad personalizadas.

INSTALACION DE JAVA

Para configurar su ambiente Java, simplemente baje la versión mas reciente de JDK del sitio Web de JavaSoft:

http://www.javasoft.com/products/jdk/ e instálelo siguiendo las instrucciones.

PAQUETE DE CODIGO EJEMPLO JAVA

Un paquete de código ejemplo está disponible en el sitio web de la facultad en: http://labsys.frc.utn.edu.ar, dirigirse a los sitios de las cátedra/PPR-2003/Unidad 1.

COMPILACION Y EJECUCION DE UN PROGRAMA

En el siguiente ejemplo se trabaja sin un entorno de desarrollo. Normalmente Ud usará uno

Vamos a describir cómo compilar y ejecutar el siguiente programa:

```
//Programa que muestra una frase en la pantalla.
import java.io.*;
public class EjemploSimple {
       public static void main(String args[]) {
              System.out.println ("Buenos Días");
       }
}
```

- Grabar el código en un archivo de texto. El programa fuente se guarda con el nombre de fichero EjemploSimple.java.
- Compilarlo en una ventana terminal (Unix) o de comandos (Windows) con "javac. EjemploSimple.*java".*

Esta instrucción provoca la compilación del fichero EjemploSimple.java, pero no crea un fichero ejecutable. Crea ficheros semicompilados, uno por clase definida en el fichero fuente (una sola en nuestro ejemplo); estos ficheros llevan el mismo nombre que la clase que describen, pero se caracterizan por la extensión .class.

- Ejecutarlo desde una ventana terminal (Unix) o de comandos (Windows) con "Java *nombre_archivo".*

  La ejecución del programa se lanza mediante el mandato:

  ```
  C: \Programasjava\UnEjemplo>java EjemploSimple
  ```

  Y éste es el resultado:

  ```
  Buenos Días
  ```

  El programa se ejecuta dentro de la Máquina Virtual Java. Por ello en lugar de :

  C: \Programasjava\UnEjemplo> EjemploSimple *<Enter>*

  Usamos

  C: \Programasjava\UnEjemplo>java EjemploSimple *<Enter>*

que lanza el núcleo Java y le indica que debe cargar la clase EjemploSimple, que encontrará en el fichero EjemploSimple.class, y ejecuta el método main() de esta clase.

Este método debe declararse public static para poder ejecutarse.

EjemploSimple.class es una librería dinámica; no contiene código ejecutable como las librerías dinámicas clásicas, sino código Java, interpretado por el núcleo.

Otro ejemplo, también muy simple

```
// segundo programa en Java
// el archivo se ha de llamar Hello.java

import java.util.Date;
class Print {
       static void prt(String s) {
              System.out.println(s);
       }
}
Public class Hello {       //el punto de entrada del programa
       public static void main(String args[]) {
              System.out.println(new Date());
              String s1 = new String("Hola mundo");
              //otra manera de inicializar cadenas
              String s2= "Hola, de nuevo";
              Print.prt(sl);
              Print.prt(s2);
       }
}
```

Vamos a estudiar detalladamente este ejemplo:

La línea ímport java.utíl.Date; avisa al compilador de que quiero utilizar la clase standard Date en mí programa. El compilador por defecto importa el package java.lang, el cual contiene clases útiles de uso general, como la clase String, usada en este ejemplo. Siguiendo con el ejemplo, vemos definida una clase Prínt, la cual sólo contiene una función estática llamada prt, la cual no retorna ningún valor y que acepta como argumentos una cadena, que será impresa por el dispositivo de salida estándar (la pantalla). Para realizar esta tarea utilizamos un objeto estático out perteneciente a la clase System (incluida en java.lang), sobre el que ejecutamos el método println().

Siguiendo con el ejemplo nos encontramos con la definición de la clase Hello, que declara y define un único método. Cuando utilizamos Java para crear una aplicación de propósito general, una de las clases de la unidad de compilación debe llamarse como el fichero y además dicha clase debe tener definida una función de la forma:

```
public static void main(String args[])
```

Esta función es el punto de entrada del programa y nos está dando a entender que esta clase puede ser usada como módulo inicial en un programa. El argumento de este método es un array de cadenas que nos sirve para poder pasar parámetros al programa mediante la línea de comandos de forma

bastante similar al C. Esta función es estática dado que va a ser llamada inicialmente al comenzar el programa, no existiendo en ese punto todavía ningún objeto creado. Esta función es pública porque tiene que poder ser accedida desde fuera del fichero. En la primera línea del cuerpo del método main nos encontramos con la sentencia System.out.println(new Date()); donde encontramos un par de ideas importantes:

- Aparece el concepto de sobrecarga de funciones que, básicamente, significa el hecho de que dos métodos tengan el mismo nombre, pero se diferencien en su lista de argumentos. En este caso el método println() está sobrecargado para poder aceptar como parámetro un objeto de tipo Date, encargándose el método de imprimir la fecha actual por la pantalla.
- Por otro lado, como parámetro de la función utilizamos un objeto creado mediante new, pero que no tiene asignado un handle. Es decir, va a ser un objeto temporal que, un cierto tiempo después de haber sido creado, será eliminado por el recolector de basura.

En las dos siguientes sentencias podemos comprobar el uso de una clase estándar para manejar cadenas constantes, llamada String. En primer lugar creamos el objeto mediante new, pasando como parámetro al constructor del objeto la cadena de caracteres que queremos manejar. El concepto de constructor será tratado ampliamente en el próximo capítulo así que, por ahora, nos basta con saber que es un método que contiene todas las clases y que facilita la inicialización de sus variables miembro. La clase String es un poco especial que nos permite usar una sintaxis similar a la del C++ para crear un objeto cadena:

```
String s2 = "Hola, de nuevo";
```

Las dos últimas sentencias del programa se limitan a llamar al método de clase prt con los argumentos apropiados (observar que no hemos necesitado crear ningún objeto Print).

EL PROCESO EN MÁS DETALLE

Los sistemas Java generalmente constan de varias partes: un entorno, el lenguaje, la interfaz de programación de aplicaciones (API, Applications Programming Interface) de Java y diversas bibliotecas de clases. A continuación analizaremos un entorno de creación de programas en Java representativo, el cual se ilustra a continuación.

Los programas Java normalmente pasan por cinco fases antes de ejecutarse. Éstas son: editar, compilar, cargar, verificar y ejecutar. En el laboratorio de sistema se utiliza el entorno JDK que sigue estrictamente estas especificaciones.

La fase 1 consiste en editar un archivo. Esto se hace con un programa editor. El programador teclea un programa en Java empleando el editor (Parte del entorno de desarrollo o independiente) y hace correcciones si es necesario. A continuación el programa se almacena en un disposifivo de almacenamiento secundario, como un disco. Los nombres de archivo de los programas en Java terminan con la extensión .java. Dos editores que se utilizan mucho en los sistemas UNIX son vi y emace (en el laboratorio se encuentran disponibles, además de estos, otros editores). Los entornos de desarrollo de Java cuentan con editores incorporados que forman parte integral del entorno de programación.

En la fase 2, el programador compila el programa. El compilador de Java traduce el programa Java a códigos de bytes, que es el lenguaje que entiende el intérprete de Java. Si desea compilar un programa llamado HolaMundo.java, teclee:

javac HolaMundo.java, (o mejor, use la opción **compile** de su entorno de desarrollo):

Si el programa se compila correctamcntc, sc producirá un archivo llamado HolaMundo.class. Éste es el archivo que contiene los códigos de bytes que serán interpretados durante la fase de ejecución.

La fase 3 se llama carga. Antes de que un programa pueda ejecutarse, es necesario colocarlo en la memoria. Esto lo hace un cargador de clases que toma el archivo (o archivos) .class que contiene los códigos de bytes y lo(s) transfiere a la memoria. El archivo .class puede cargarse de un disco de su sistema o a través de una red. El cargador de clases comienza a cargar archivos .class en dos situaciones. Por ejemplo, el comando:

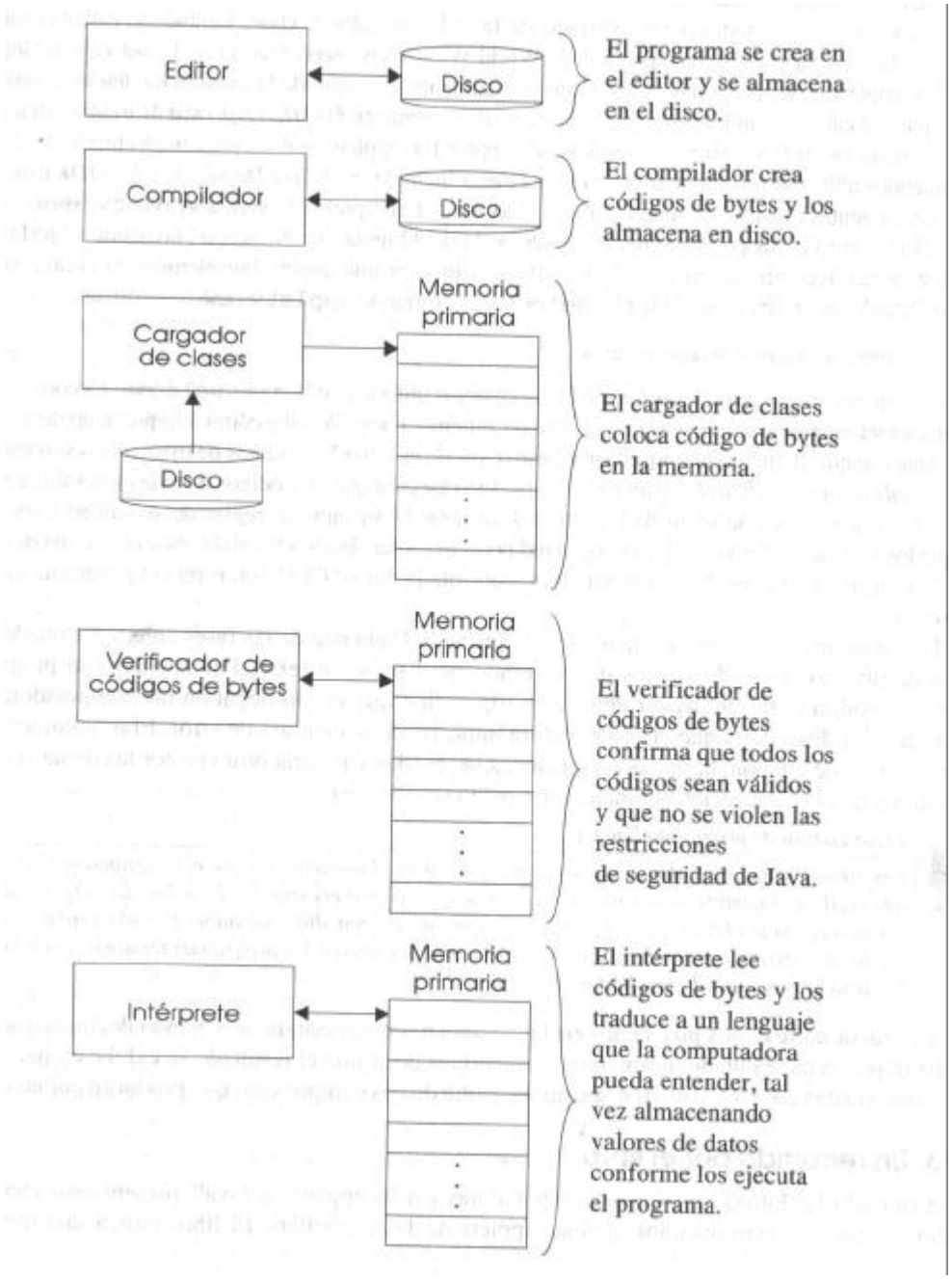
java HolaMundo, (o mejor, use la opción **run** de su entorno de desarrollo):

Invoca el intérprete java para el programa HolaMundo y hace que el cargador de clases, cargue la información empleada en el programa HolaMundo. Llamamos al programa HolaMundo una aplicación. Las aplicaciones son programas que son ejecutados por el intérprete(Máquina virtual) de Java. El cargador de clases también se ejecuta cuando una applet de Java se carga en un navegador de la World Wide Web, como Navigator de Netscape o Explorer de Microsoft.

Antes de que el intérprete pueda ejecutar los códigos de bytes, éstos son verificados por el verificador de códigos de bytes en la fase 4. Esto asegura que los códigos de bytes son válidos y que no violan las restricciones de seguridad de Java. Java debe hacer cumplir reglas de seguridad muy estrictas porque los programas Java que llegan por la red podrían causar daños a los archivos y el sistema del usuario.

Por último, en la fase 5, la computadora, controlada por su CPU, interpreta del programa, un código de byte a la vez.

Los programas casi nunca funcionan a la primera. Cada una de las fases anteriores puede fallar a causa de diversos tipos de errores. El programador regresaría a la fase de edición, haría las correcciones necesarias y pasaría otra vez por las demás fases para determinar que las correcciones funcionaron como se esperaba.



PROGRAMACION ORIENTADA A OBJETOS

INTRODUCCION a CLASES

Como ya dijimos, una clase es un modelo que se utiliza para describir a objetos similares.

Las clases son lo más simple de Java. Todo en Java forma parte de una clase, es una clase o describe como funciona una clase. El conocimiento de las clases es fundamental para poder entender los programas Java.

Todas las acciones de los programas Java se colocan dentro del bloque de una clase o un objeto. Todos los métodos se definen dentro del bloque de la clase, Java no soporta funciones o variables globales. Esto puede despistar a los programadores de C++, que pueden definir métodos fuera del bloque de la clase, pero esta posibilidad es más un intento de no separarse mucho y ser compatible con C, que un buen diseño orientado a objetos. Así pues, el esqueleto de cualquier aplicación Java se basa en la definición de una clase.

Todos los datos básicos, como los enteros, se deben declarar en las clases antes de hacer uso de ellos. En C la unidad fundamental son los archivos con código fuente, en Java son las clases. De hecho son pocas las sentencias que se pueden colocar fuera del bloque de una clase. La palabra clave import (equivalente al #include) puede colocarse al principio de un archivo, fuera del bloque de la clase. Sin

embargo, el compilador reemplazará esa sentencia con el contenido del archivo que se indique, que consistirá, como es de suponer, en más clases.

Una clase es un marco que permite crear objetos de su misma estructura. Estos objetos constan de:

- Variables de clase y de instancia, que son los descriptores de atributos y entidades de los objetos.
- Métodos, que definen las operaciones que pueden realizar esos objetos.

Un objeto se instancia a partir de la descripción de una clase: un objeto es dinámico, y puede tener una infinidad de objetos descritos por una misma clase. Un objeto es un conjunto de datos presente en memoria.

ESTRUCTURA GENERAL

Una clase contiene elementos, llamados miembros, que pueden ser datos, llamados *atributos,* y funciones que manipulan esos datos llamados *métodos*.

Una clase se define con la palabra reservada class.
La sintaxis de una clase es:

```
[public] [final | abstract] class nombre_de_la_Clase [extends ClaseMadre]
      [implements Interfase1 [, Interfase2 ]...]
{
      [Lista_de_atributos]
      [lista_de_métodos]
}
```

Todo lo que está entre [y] es opcional. Como se ve, lo único obligatorio es class y el nombre de la clase.

public, final, abstract

Definir una clase como pública (**public**) significa que puede ser usada por cualquier clase en cualquier paquete. Si no lo es, solamente puede ser utilizada por clases del mismo paquete (un paquete, básicamente, es un grupo de clases e interfaces relacionadas, como los paquetes de biblioteca incluidos con Java).

Una clase final (**final**) es aquella que no puede tener clases que la hereden. Esto se utiliza básicamente por razones de seguridad (para que una clase no pueda ser reemplazada por otra que la herede), o por diseño de la aplicación.

Una clase abstracta (**abstract**) es una clase que puede tener derivadas, pero no puede ser instanciada. Es literalmente abstracta. ¿Para qué sirve? Para modelar conceptos. Por ejemplo, la clase Number es una clase abstracta que representa cualquier tipo de números (y sus métodos no están implementados: son abstractos); las clases descendientes de ésta, como Integer o Float, sí implementan los métodos de la madre Number, y se pueden instanciar.

Por todo lo dicho, una clase no puede ser final y abstract a la vez (ya que la clase abstract requiere descendientes).

extends

La instrucción extends indica de qué clase desciende la nuestra. Si se omite, Java asume que desciende de la superclase **object**.

Cuando una clase desciende de otra, significa que hereda sus atributos y sus métodos. Esto quiere decir que, a menos que los redefinamos, sus métodos serán los mismos que los de la clase madre y podrán utilizarse en forma transparente, siempre y cuando no sean *privados* en la clase madre, o protegidos o propios del paquete para subclases de otros paquetes.

DECLARACION Y DEFINICION

La *declaración* lista los miembros de la clase. La definición, también llamada *implementación*, define las funciones de la clase.

La declaración y la definición de la clase van juntas. Por ejemplo:

```
//Implementación de un contador sencillo
//GRABAR EN UN ARCHIVO "Contador.java"
// COMPILAR CON: "javac Contador.Java
public class Contador {    //     Se declara y define la clase Contador
       int cnt;
```

```
                public void Inicializa() {
                cnt=0;       //inicializa en 0 la variable cnt
        }
        //Otros métodos
        public int incCuenta() {
                cnt++;
                return cnt;
        }
        public int getCuenta() {
                return cnt;
        }
}
```

EL CUERPO DE LA CLASE

El cuerpo de la clase, encerrado entre { y }, es la lista de atributos (variables) y métodos (funciones) que la constituyen. No es obligatorio, pero en general se listan primero los atributos y luego los métodos.

INSTANCIAS DE UNA CLASE (OBJETOS)

Los objetos de una clase son instancias de la misma. Se crean en tiempo de ejecución con la estructura definida en la clase.

Para crear un objeto de una clase se usa la palabra reservada new.

Por ejemplo si tenemos la siguiente clase:

```
public class Cliente {
        private int codigo;
        private float importe;
        public int   getCodigo()   { return codigo; }
        public float getImporte()  { return importe; }
        public void setImporte(float x) { importe = x; }
};
```

el objeto o instancia de la clase cliente es:

```
Cliente comprador = new Cliente();      //objeto o instancia de la clase
                                        //Cliente
```

El operador new reserva espacio en memoria para los miembros dato y devuelve una referencia que se guarda en la variable comprador de tipo Cliente que denominamos ahora objeto. Dicha sentencia, crea un objeto denominado comprador de la clase Cliente.

Implementando el ejemplo en forma completa:

```
import java.io.*;

Public class ManejaCliente {
        //el punto de entrada del programa
        public static void main(String args[]) {
                Cliente comprador = new Cliente();      //crea un cliente

                comprador.setImporte(100);       //asigna el importe 100

                float adeuda = comprador.getImporte();
                System.out.println("El importe adeudado es "+adeuda);
        }
}
```

Acceso a los miembros

Desde un objeto se puede acceder a los miembros mediante la siguiente sintaxis

```
objeto.miembro;
```

Por ejemplo, podemos acceder al método setImporte, para cambiar el importe de la deuda de un objeto cliente.

```
comprador.setImporte(100);
```

Si el cliente comprador, por ejemplo, tenía inicialmente un importe de 0, mediante esta sentencia se lo cambiamos a 100.

Desde un objeto llamamos a las funciones miembro para realizar una determinada tarea. Por ejemplo, desde el cliente comprador llamamos a la función getImporte() para obtener el importe de dicho cliente.

```
Comprador.getImporte();
```

La función miembro getImporte() devuelve un número, que guardaremos en una variable adeuda, para luego usar este dato.

```
float adeuda=comprador.getImporte();
System.out.println("El importe adeudado es "+adeuda);
```

Como veremos mas adelante, no siempre es posible acceder a los miembros, el acceso dependerá de los controles de acceso a los mismos.

Ciclo de Vida de los Objetos

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos.

1. Los objetos se crean a medida que se necesitan.
2. Los mensajes se mueven de un objeto a otro (o del usuario a un objeto) a medida que el programa procesa información o responde a la entrada del usuario.
3. Cuando los objetos ya no se necesitan, se borran y se libera la memoria.

La vida de un objeto

En el lenguaje C++, los objetos que se crean con new se han de eliminar con delete. new reserva espacio en memoria para el objeto y delete libera dicha memoria. En el lenguaje Java no es necesario liberar la memoria reservada, el recolector de basura (garbage collector) se encarga de hacerlo por nosotros, liberando al programador de una de las tareas que más quebraderos de cabeza le producen, olvidarse de liberar la memoria reservada.

DECLARACION DE MIEMBROS UNA CLASE

Los datos de una clase se denominan **atributos** y las funciones de una clase se denominan **métodos**.

Los miembros tienen ciertas restricciones en el modo en que se puede manipular los mismos dentro y fuera de la clase, a esto se le llama control de acceso a una clase o visibilidad.

Con estas restricciones se logra la encapsulación que, como vimos en la introducción, consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles de implementación del mismo, que quedan ocultos para los demás. La encapsulación se basa en la noción de servicios prestados; Una clase proporciona un cierto número de servicios y los usuarios de esta clase no tienen que conocer la forma como se prestan estos servicios. Contrariamente a las cuentas de una asociación caritativa, una clase debe mantenerse opaca.

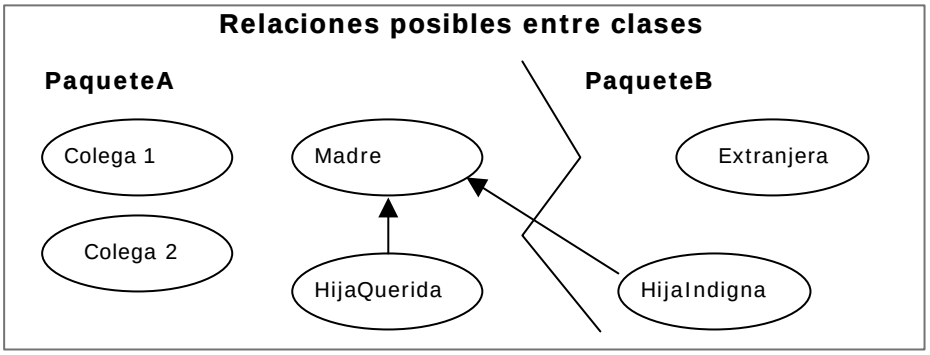
Hay que distinguir pues en la descripción de la clase dos partes:

- la parte pública, accesible por las otras clases;
- la parte privada, accesible únicamente por los métodos de la clase.

Se recomienda encarecidamente poner los atributos de una clase en la parte privada, para respetar el concepto de encapsulamiento.

Modificadores de acceso a miembros de clases

Java proporciona varios niveles de encapsulamiento que vamos a examinar a continuación.



Hemos representado en este dibujo una clase Madre alrededor de la cual gravitan otras clases:

- sus hijas **HijaQuerida e HijaIndigna**, la segunda de las cuales se encuentra en otro paquete; estas dos clases heredan de Madre. la herencia se explica en detalle algo más adelante, pero retenga que las clases HijaQuerida e HijaIndigna se parecen mucho a la clase Madre. Los paquetes se detallan igualmente algo más adelante; retenga que un paquete o *package* es un conjunto de clases relacionadas con un mismo tema destinada para su uso por terceros, de manera análoga a como otros lenguajes utilizan las librerías.

- sus **colegas**, que no tienen relación de parentesco pero están en el mismo paquete;

- una clase **Extranjera**, sin ninguna relación con la clase Madre.

En el esquema anterior, así como en los siguientes, la flecha simboliza la relación de herencia.

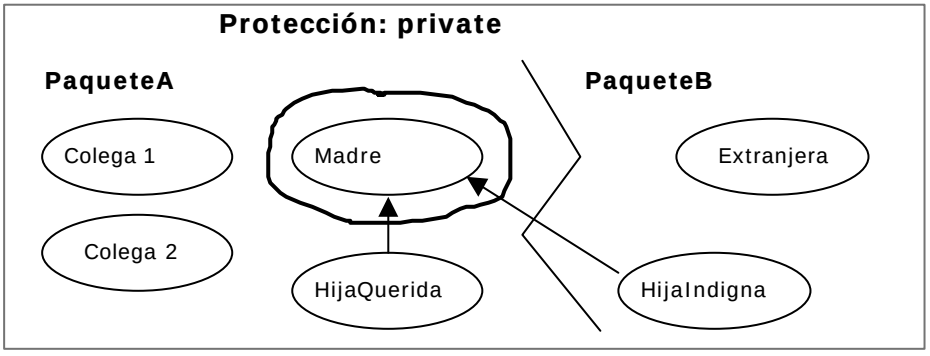
En los párrafos siguientes vamos a examinar sucesivamente los diferentes tipos de protección que Java ofrece a los atributos y a los métodos. Observemos ya desde ahora que hay cuatro tipos de protección posibles y que, en todos los casos, la protección va sintácticamente al principio de definición, como en los dos ejemplos siguientes, donde private y public definen los niveles de protección:

```
private void Metodo ();
public int Atributo;
```

Private

La protección más fuerte que puede dar a los atributos o a un método es la protección **private**.

Esta protección impide a los objetos de otras clases acceder a los atributos o a los métodos de la clase considerada. En el dibujo siguiente, un muro rodea la clase Madre e impide a las otras clases acceder a aquellos de sus atributos o métodos declarados como **private**.



Insistimos en el hecho de que la protección no se aplica a la clase globalmente, sino a algunos de sus atributos o métodos, según la sintaxis siguiente:

```
private void MetodoMuyProtegido () {
      //    ...
}
private int AtributoMuyProtegido;
public int OtraCosa;
```

Observe que un objeto que haya surgido de la misma clase puede acceder a los atributos privados, como en el ejemplo siguiente:

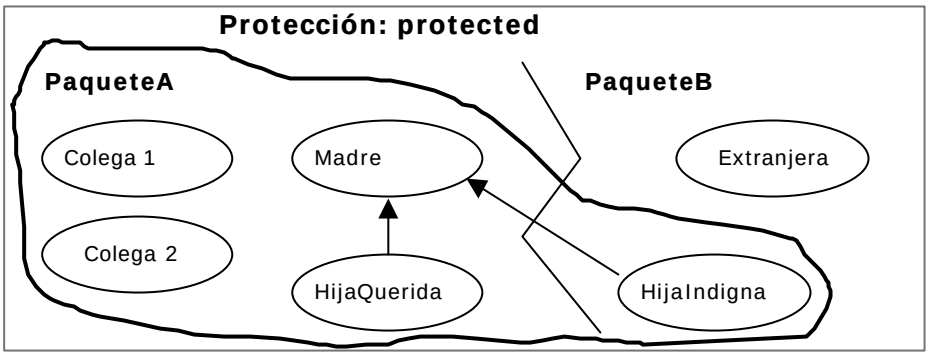
```
class Secreta {
      private int s;
      void init (Secreta otra) {
            s = otra.s;
      }
```

```
}
```

La protección se aplica pues a las relaciones entre clases y no a las relaciones entre objetos de la misma clase.

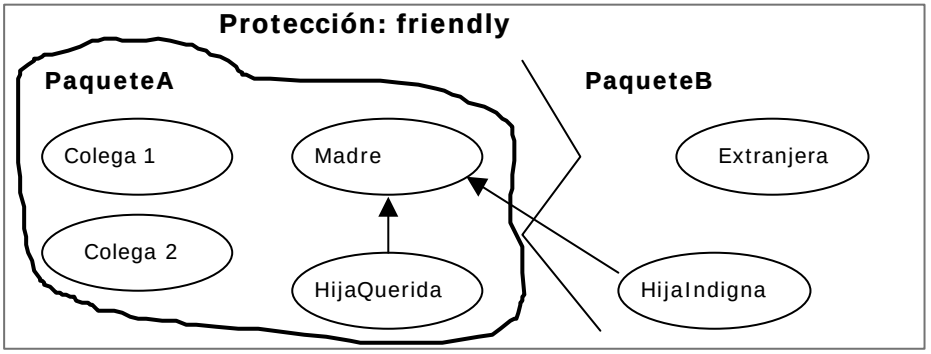
Protected

El tipo de protección siguiente viene definido por la palabra clave **protected** simplemente. Permite restringir el acceso a las subclases y a las clases del mismo paquete.



Friendly

El tipo de protección predeterminado se llama **friendly.** En Java si no se indica explícitamente ningún nivel de protección para un atributo o un método, el compilador considera que lo ha declarado friendly. No tiene por qué indicar esta protección explícitamente. Además, friendly no es una palabra clave reconocida. Esta protección **autoriza el acceso a las clases del mismo paquete**, pero no incluye a las subclases.



Los miembros con este tipo de acceso pueden ser accedidos por todas las clases que se encuentren en el paquete donde se encuentre también definida la clase.

Para recordar:

- Friendly es el tipo de protección asumido por defecto.
- Un paquete se define por la palabra reservada package.
- Un paquete puede ser nominado o no.
- Las clases se codifican en archivos .java
- Varios archivos pueden pertenecer al mismo paquete.
- Un archivo solo pertenece a un paquete. Solo se permite una cláusula package por archivo.
- Los archivos que no tienen cláusulas package pertenecen al paquete unnamed.

Public

El último tipo de protección es public. Un atributo o un método calificado así es accesible para todo el mundo.

¿Un caso excepcional? No. Muchas clases son universales y proporcionan servicios al exterior de su paquete: las librerías matemáticas, gráficas, de sistema, etc., están destinadas a ser utilizadas por

cualquier clase. Por el contrario, una parte de estas clases está protegida, a fin de garantizar la integridad del objeto.

Se suele decir que los miembros públicos conforman la interfaz de la clase con el usuario.

Separación de la interfaz

¿Cuándo debe utilizarse qué? O en otras palabras: ¿cuáles son los diferentes casos de utilización de los mecanismos de protección? Nos proponemos distinguir principalmente dos casos:

- el atributo o el método pertenece a la *interfaz* de la clase: debe ser public;
- el atributo o la clase pertenece al *cuerpo* de la clase: debe ser protegido. Esta protección es diferente según los casos; en general, la protección más fuerte es aconsejable porque es fácil desproteger un atributo, y es mucho más difícil hacerlo inaccesible si ya se utiliza.

La **interfaz** de la que hablamos es pues la interfaz conceptual de la clase, es decir los atributos y las signaturas de los métodos (tipo devuelto + nombre + parámetros), directamente utilizados desde el exterior porque corresponden a un servicio prestado.

El **cuerpo** de la clase es la implementación de dicho servicio.

Es decir, la interfaz de la clase es el ***qué*** -qué hace la clase-, mientras que su cuerpo es el ***cómo*** -cómo lo hace.

Por esto conviene hacer pública la interfaz y proteger el cuerpo.

Ahora, veamos un ejemplo en pseudocódigo sobre cómo crear un objeto reloj que demuestre cuáles serían sus funciones públicas y sus funciones privadas:

Función inicial o Constructor:
Reloj negro, hora inicial 12:00am;
Funciones Públicas:
Apagar
Encender
Poner despertador;
Funciones Privadas:
Mecanismo interno de control
Mecanismo interno de baterías
Mecanismo de manecillas

Al utilizar uno de estos relojes nos importa su operación no su mecanismo interno, por eso existen funciones públicas o privadas. Las funciones públicas son la interfaz que usaremos. El constructor inicializa al objeto en un estado inicial estable para su operación.

Un ejemplo más, esta vez con una computadora:

Función inicial o Constructor:
Computadora portátil compaq, sistema operativo windows98, encendida
Funciones Públicas:
Apagado
Teclado
Pantalla
Impresora
Bocinas
Funciones Privadas:
Caché del sistema
Procesador
Dispositivo de Almacenamiento
Motherboard

Obviamente si se abre (físicamente) la computadora se tendrá acceso a todo, sucede lo mismo si se abre el código fuente del archivo de una clase, se puede modificar todo, pero debe quedar bien claro que la razón por la cual se hace todo lo anterior es para organizar mejor el código, no para impedir el acceso a nadie a ciertas cosas, todo es para mejorar la operación general de una clase ocultando la información que no es necesario que conozcamos y exponiendo la que sí.

ATRIBUTOS DE UNA CLASE

Todas las variables se declaran dentro del cuerpo de la clase o dentro de un método.

- ✓ Las variables declaradas dentro de un método son ***locales*** a él;
- ✓ las variables declaradas en el cuerpo de la clase se dice que son ***miembros*** de ella y son accesibles por todos los métodos de la clase.

Por otra parte, además de los atributos de la propia clase, se puede acceder a todos los atributos de la clase de la cual desciende. Por ejemplo, cualquier clase que descienda de la clase **Polygon** hereda los atributos **npoints, xpoints** e **ypoints**. Esto lo veremos con mayor detalle cuando estudiemos herencia.
Finalmente, los atributos miembros de la clase pueden ser *atributos de clase* o *atributos de instancia;*

- ✓ se dice que son atributos *de clase* si se usa la palabra clave ***static***: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria).
- ✓ Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (o sea que es independiente para cada objeto).

Los atributos pueden ser:

- ✓ tipos básicos. (no son clases)
- ✓ clases e interfases (clases de tipos básicos, clases propias, de terceros, etc.)

La lista de atributos sigue el mismo formato de C: se define primero el tipo y luego el nombre del atributo y, finalmente, un ";". Por ejemplo:

```
int n_entero;
float n_real;
char p;
```

La declaración sigue siempre el mismo esquema:

[**Modificador de acceso**] [**static**] [**final**] [**transient**] [**volatile**] **Tipo NombreVariable [= Valor];**

El modificador de acceso puede ser alguno de los que vimos anteriormente (private, protected, public, etc).

static sirve para definir un atributo como de clase, o sea, único para todos los objetos de ella.

final, como en las clases, determina que un atributo no pueda ser sobreescrito o redefinido, es decir que no se trata de una variable, sino de una *constante.*

transient denomina atributos que no se graban cuando se archiva un objeto, o sea que no forman parte del estado permanente de éste.

volatile se utiliza con variables modificadas en forma asincrónica por objetos en diferentes *threads* (literalmente "hilos", tareas que se ejecutan en paralelo). Básicamente, esto implica que distintas tareas pueden intentar modificar la variable de manera simultánea, y volatile asegura que se vuelva a leer la variable (por si fue modificada) cada vez que se la va a usar.

Atributos estáticos de una clase

Le hemos explicado anteriormente que cada objeto poseía sus propios atributos. Es posible que todos los objetos de una misma clase tengan atributos en común: son los atributos de clase, introducidos por la palabra clave static. Estos atributos son legibles y modificables por todos los objetos de una misma clase. La modificación de un atributo static es tenida en cuenta inmediatamente por los otros objetos, porque lo *comparten.*

Un miembro de una clase se puede declarar estático (static). Para un miembro dato, la designación static significa que existe sólo una instancia de ese miembro en la clase. Un miembro dato estático es compartido por todos los objetos de una clase y existe incluso si ningún objeto de esta clase existe siendo su valor común a la clase completa.

A un miembro dato static se le asigna una zona fija de almacenamiento en tiempo de compilación, al igual que una variable global, pero el identificador de la variable está dentro del ámbito utilizando solamente el operador de resolución con el nombre de la clase.

Ejemplo:

```
import java.io.*

class Participante {
        static int participado = 2;
        int noparticipado = 2;
        void Modifica () {
                participado = 3;
                noparticipado = 3;
        }
}

class demostatic {
        static public void main (String [] arg) {
                Participante p1 = new Participante ();
                Participante p2 = new Participante ();
```

```
                        System.out.println("pl: " + pl.participado + " " +
                              pl.noparticipado);
                        pl.Modifica ();
                        System.out.println("pl: " + pl.participado + " " +
                              pl.noparticipado);
                        System.out.println("p2: " + p2.participado + " " +
                              p2.noparticipado);
                  }
            }
```

dará como resultado:

```
C:\Programasjava\objetos>java demostatic
pl: 2 2
pl: 3 3
p2: 3 2
```

En efecto, la llamada a Modifica () ha modificado el atributo participado. Este resultado se extiende a todos los objetos de la misma clase, mientras que sólo ha modificado el atributo noparticipado del objeto actual.

METODOS DE UNA CLASE

Un método es una función que se ejecuta sobre un objeto. No se puede ejecutar un método sin precisar el objeto sobre el que se aplica (salvo una excepción que veremos más adelante).

Los métodos de una clase definen las operaciones que un usuario puede realizar con los atributos de la clase. Desde el punto de vista de la POO, el conjunto de todas las funciones definen el conjunto de mensajes a los que los objetos de las citadas clases pueden responder.

Declaración y definición

Los métodos, como las clases, tienen una declaración y un cuerpo. La declaración es del tipo:

[**modificador de acceso**] [**static**] [**abstract**] [**final**] [**native**] [**synchronized**] TipoDevuelto NombreMétodo (tipo1 nombre1 [**,** tipo2 nómbre2]...]) [**throws** excepción [**,** excepción2].

La declaración y definición se realizan juntas, es decir en el cuerpo de la clase. Básicamente, los métodos son como las funciones de C: implementan el cálculo de algún parámetro (que es el que devuelven al método que los llama) a través de funciones, operaciones y estructuras de control. Sólo pueden devolver un valor (del tipo TipoDevuelto), aunque pueden no devolver ninguno (en ese caso TipoDevuelto es void). El valor de retorno se especifica con la instrucción return, dentro del método.

Los métodos pueden utilizar valores que les pasa el objeto que los llama *(parámetros),* indicados con tipo1 nombre1, tipo2 nombre2... en el esquema de la declaración. Estos parámetros pueden ser de cualquiera de los tipos válidos. Si son tipos básicos, el método recibe el *valor* del parámetro; si son arreglos, clases o interfaces, recibe un puntero a los datos *(referencia).* Veamos un pequeño ejemplo:

```
Public int AumentarCuenta(int cantidad) {
      cnt = cnt + cantidad;
      return cnt;
}
```

Este método, si lo agregamos a la clase Contador, le suma cantidad al acumulador cnt. En detalle:

- el método recibe un valor entero (cantidad).
- lo suma a la variable de instancia cnt.
- devuelve la suma (return cnt).

El **modificador de acceso** puede ser alguno de los que vimos anterioremente (private, protected, public, etc).

El resto de la declaración

Los métodos estáticos (**static**) son, como los atributos, métodos *de clase:* si el método no es static, es un método *de instancia.* El significado es el mismo que para los atributos: un método static es compartido por todas las instancias de la clase.

Los métodos abstractos (**abstract**) son aquellos de los que se da la declaración pero no la implementación (o sea que consiste sólo en el encabezamiento). Cualquier clase que contenga al menos un método abstracto (o cuya clase madre contenga al menos un método abstracto que no esté implementado en la hija) es una clase abstracta.

Un método es final (**final**) cuando no puede ser redefinido por ningún descendiente de la clase.

Los métodos **native** son aquellos que se implementan en otro lenguaje propio de la máquina (por ejemplo, C o C++). Sun aconseja utilizarlas bajo riesgo propio, ya que, en realidad, son ajenas al lenguaje. Pero existe la posibilidad de usar viejas bibliotecas que uno armó y no tiene ganas de reescribir, ¡a costa de perder portabilidad!

Los métodos **synchronized** permiten sincronizar varios *threads* para el caso en que dos o más accedan concurrentemente a los mismos datos.

Finalmente, la cláusula **throws** sirve para indicar que la clase genera determinadas excepciones.

El cuerpo de los métodos

En Java dentro de los métodos pueden incluirse:

- Declaración de variables locales
- Asignaciones a variables
- Operaciones matemáticas
- Llamados a otros métodos
- Estructuras de control
- Excepciones (try, catch, que veremos más adelante)

Declaración de variables locales

Las variables locales en un método se declaran igual que en C:

```
Tipo NombreVariable [ = Valor];
```

Por ejemplo:

```
int    suma;
float  precio;
Contador laCuenta;
```

Las variables pueden inicializarse al momento de su creación:

```
int    suma=0;
float  precio = 12.3;
Contador laCuenta = new Contador() ;
```

Llamadas a métodos

Se llama a un método de la misma clase simplemente con el nombre del método y los parámetros entre paréntesis, como se ve, entre otros, en el ejemplo siguiente:

Otra vez recordaremos la clase Contador:

```
//Implementación de un contador sencillo
public class Contador {    //     Se declara y define la clase Contador
       int cnt;
       public void Inicializa() {
              cnt=0;       //inicializa
       }
       //Otros métodos
       public int incCuenta() {
              cnt++;
              return cnt;
       }
       public int getCuenta() {
                     return cnt;
       }
}
```

Pueden probar esta clase (mínima) con el siguiente ejemplo de aplicación:

```
import java.io.*;

public class Ejemplollamadas {
       public static void main(String args[]) {
              Contador c = new Contador;
              c.Inicializa();
              System.out.println(c.incCuenta());
              System.out.println(c.getCuenta());
```

```
            }
        }
```

las líneas en letra resaltada es un ejemplo de un llamado a un método de un objeto de otra clase. Noten que en este caso, es necesario llamar al método para un objeto (instancia) existente, por lo que se indica:

Nombre_del_Objeto<punto>Nombre_del_método(parámetros)

El objeto actual (puntero this)

Nunca se puede llamar una función miembro de una clase a menos que se asocie con un objeto (una instancia de la clase). ¿Cómo sabe una función miembro cuál es la instancia de una clase (el objeto específico) asociada con ella?.

El método utilizado por Java es añadir un argumento extra oculto a las funciones miembro. Este argumento es un puntero al objeto de la clase que los enlaza con la función asociada y recibe un nombre especial denominado **this**.

Dentro de una función miembro, this apunta al objeto asociado con la invocación de la función miembro. Normalmente, el programador no necesita preocuparse por este puntero, ya que el lenguaje realiza la operación automáticamente transparente a las funciones miembro que la utilizan.

Las referencias a los miembros del objeto asociado a la función se realiza con el prefijo this y el operador de acceso punto **.** .

Si tomamos como ejemplo la siguiente clase:

```
public class Cliente {
        private int codigo;
        private float importe;
        public int   getCodigo()   { return codigo; }
        public float getImporte()  { return importe; }
        public void setImporte(float x) { importe = x; }
};
```

Cuando escribimos en el método setImporte

```
importe = x;
```

para asignar un valor a importe, no tuvimos necesidad de indicar a qué objeto pertenecía. Cuando no se pone el objeto antes del atributo, se asume que la variable es un miembro del objeto, o es una variable local o parámetro. Si los nombres colisionan, como podría ser en el siguiente método

```
public void setImporte(float importe)
{
        this.importe = importe;
}
```

usamos this para indicar al objeto actual. Esta palabra reservada siempre está apuntando al objeto actual, y puede usarse como en este caso, para resolver una ambigüedad, o puede usarse, por ejemplo, como parámetro en una llamada a una función para pasar un puntero al objeto asociado:

```
objeto.procesar(this);
```

Métodos especiales

Métodos sobrecargados

Tradicionalmente, un método o una función realizan una tarea específica para la que están programados. Java soporta la *sobrecarga de métodos,* lo que le permite definir versiones de éstos con el mismo nombre en una clase, siempre y cuando las versiones tengan diferentes firmas. Una firma incluye el nombre del método, el número, el orden y los tipos de sus parámetros formales. Como ejemplo simple, considere el reestablecímiento del saldo de cuenta de un objeto Account, esta clase (Account) ya tiene un método

```
public double balanceo()
{
        return saldo;
}
```

que se utiliza para recuperar el saldo de la cuenta. Con la sobrecarga podemos definir un método:

```
public void balanceo(double valor)
{
        saldo = valor;
}
```

que fija el saldo en una cantidad específica. El mismo método recuperará o establecerá el saldo, dependiendo de que se proporcione un argumento; ¡es muy útil!

Tenga en cuenta que, sólo ocurre la sobrecarga, cuando se utiliza varias veces el mismo nombre de método dentro de una clase. No hay un límite práctico para las versiones que es posible apilar en el mismo nombre de método.

Resolución de llamada a un método

Cuando se hace una llamada aun método sobrecargado Java deduce automáticamente, a partir de los argumentos reales, la versión correcta del método que habrá de invocar. A esta actividad se le denomina *resolución de llamada,* Java la realiza al seleccionar un método entre los accesibles *que son aplicables.*

Un método es *aplicable* si toma el mismo número de parámetros que los argumentos dados y cada uno de éstos puede transformarse por *conversión de invocación de método* al tipo del parámetro.

El compilador realiza la resolución de llamada de método al comparar el número y tipo de los argumentos reales con firmas de todos los métodos accesibles y elige un método aplicable que es el *más* especifico.

En resumen, las funciones sobrecargadas tienen el mismo nombre, pero deben tener un número diferente de argumentos o diferentes tipos de argumentos, o ambos. Por ejemplo:

```
void visualizar();
void visualizar(int cuenta);
void visualizar(int cuenta, int max);
```

Ejemplo:

Supongamos que tenemos una clase Media, que calcula la media de dos y tres números, enteros y reales. Para esto tendremos las siguientes funciones:

```
float media (float, float);       //calcula la media de dos valores tipo float
int media (int, int);             //calcula la media de dos valores tipo int
float media (float, float, float); //calcula la media de tres valores tipo float
int media (int, int, int);        // calcula la media de tres valores tipo float
```

Entonces:

```
public class Media {
       public float Cal_Media (float a, float b)
       { return (a+b)/2.0; }
       public int Cal_Media (int a, int b)
              { return (a+b)/2; }
       public float Cal_Media (float a, float b, float c)
       { return (a+b+c)/3.0;}
       public int Cal_Media (int a, int b, int c)
       { return (a+b+c)/3; }
};

public class demoMedia {
       public static void main (String arg[]) {
              Media M = new Media();
              float x1, x2, x3;
              int y1, y2, y3;
              ...
                     System.out.println(M.Cal_Media (x1, x2));
                     System.out.println(M.Cal_Media (x1, x2, x3));
                     System.out.println(M.Cal_Media (y1, y2));
                     System.out.println(M.Cal_Media (y1, y2, y3));
       }
}
```

Métodos constructores

Para cada clase, pueden definirse uno o más métodos particulares: son los constructores.
Un constructor es una función especial que sirve para construir o inicializar objetos.
En general:

- Tienen el mismo nombre de la clase que inicializa.
- No devuelven valores.
- Pueden admitir parámetros como cualquier otra función.
- Pueden existir más de un constructor, e incluso no existir.
- Si no se define ningún constructor de una clase, el compilador generará un constructor por defecto.
- Se llaman en el momento de la creación del objeto.

```
//Implementación de un contador sencillo
//GRABAR EN UN ARCHIVO "Contador.java"
// COMPILAR CON: "javac Contador.Java
public class Contador {    //     Se declara y define la clase Contador
       int cnt;
       public Contador() {
              cnt = 0;            //inicializa en 0
       }
       public Contador(int c) {
              cnt = c;            //inicializa con el valor de c
       }
       //Otros métodos
       public int getCuenta() {   return cnt;}
       public int incCuenta() {   cnt++;return cnt;
       }
}
```

Pueden probar esta clase (mínima) con el siguiente ejemplo de aplicación:

```
// Archivo:  EjemploConstructor.java
//Compilar con: javac EjemploConstructor.java
//Ejecutar con: java EjemploConstructor
import java.io.*;

public class EjemploConstructor {
       public static void main(String args[]) {
              Contador c1 = new Contador;
              Contador c2 = new Contador(20);
              System.out.println(c1.getCuenta());
              System.out.println(c2.getCuenta());
       }
}
```

Cuando, desde una aplicación u otro objeto, se crea una instancia de la clase Contador, mediante la instrucción:

```
Contador c1 = New Contador();
```

el compilador busca un método con el mismo nombre de la clase y que se corresponda con la llamada en cuanto al tipo y número de parámetros. Dicho método se llama Constructor, y una clase puede tener más de un constructor (no así un objeto instanciado, ya que, una vez que fue creado, **no** puede "recrearse" a sí mismo).

En tiempo de compilación o ejecución, según corresponda, al encontrarse dicha instrucción, se reserva espacio para el objeto instanciado y se crea su estructura y en tiempo de ejecución se llama al método constructor.

La utilización de las instrucciones vistas, por ejemplo new(), entraña en efecto la creación física del objeto y la llamada a uno de sus constructores. Si hay varios constructores, estos difieren unos de otros por los parámetros que se les pasa mediante new().

Por ejemplo:

```
import java.io.*;
// una clase que tiene dos constructores diferentes
class Ejemplo {
       public Ejemplo (int param) {
              System.out.println ("Ha llamado al constructor");
              System.out.println ("con un parámetro entero");
       }
       public Ejemplo (String param) {
              System.out.println ("Ha llamado al constructor'.);
              System.out.println ("con un parámetro String");
       }
}

// una clase que sirve de main
public class democonstructor {
       public static void main (String arg[]) {
```

```
                Ejemplo e;
                e = new Ejemplo (2);
                e = new Ejemplo ("2");
        }
}
```

da el resultado siguiente:

```
c:\Programasjava\objetos>java democonstructor
Ha llamado al constructor con un parámetro entero
Ha llamado al constructor con un parámetro String
```

Los constructores no tienen tipo de retorno. Atención, si por error definimos un constructor que tenga un tipo de retomo, el compilador lo considerará como un método normal. En ese caso tendremos la impresión de que el constructor no se llama en el momento de la creación del objeto. En realidad, se llamará a un constructor predeterminado, porque no habremos definido realmente un constructor.

Tipos de constructores

Constructores por defecto

El constructor por defecto es un constructor que no acepta argumentos. Por ejemplo:

```
class Punto {
        int x;
        int y;
        public Punto()
        {       x = 0;
                y = 0;
        }
}
```

Para crear objetos usando este constructor se escribiría:

```
public class constructorPorDefecto {
        public static void main (String arg[]) {
                Punto p1;
                p1 = new Punto();
        }
}
```

Constructores con argumentos

El constructor con argumentos, como su nombre lo indica posee argumentos. Por ejemplo:

```
class Punto {
        int x;
        int y;
        public Punto(int x, int y)
        {       this.x = x;
                this.y = y;
        }
}
```

Para crear objetos usando este constructor se escribiría:

```
public class constructorArgumentos {
        public static void main (String arg[]) {
                Punto p2;
                p2 = new Punto(2, 4);
        }
}
```

Constructores copiadores

Un constructor que crea un objeto a partir de uno existente se llama constructor copiador o de copias. Es un constructor que toma como único parámetro otro objeto del mismo tipo. El constructor de copia tiene sólo un argumento: una referencia a un objeto de la misma clase. Por ejemplo:

```
class Punto {
        int x;
        int y;
```

```
        public Punto(Punto p)
        {       x = p.x;
                y = p.y;
        }
}
```

Para crear objetos usando este constructor se escribiría:

```
public class constructorCopia {
        public static void main (String arg[]) {
                Punto p2;                   //del ejemplo anterior
                p2 = new Punto(2, 4);

                Punto p3;
                p3 = new Punto(p2);         //p2 sería el objeto creado en el
                                            //ejemplo anterior
        }
}
```

o bien

```
//p2 sería el objeto creado en el ejemplo anterior
public class constructorCopia {
        public static void main (String arg[]) {
                Punto p2;                   //del ejemplo anterior
                p2 = new Punto(2, 4);

                Punto p3;
                p3 = p2;              //p2 sería el objeto creado en el
                                      //ejemplo anterior
        }
}
```

En esta asignación no se crea ningún objeto, solamente se hace que p2 y p3 apunten y referencien al mismo objeto.

Caso especial

En Java es posible inicializar los atributos indicando los valores que se les darán en la creación del objeto. Estos valores se adjudican tras la creación física del objeto, pero antes de la llamada al constructor.

```
import java.io.*
class Reserva {
        int capacidad = 2;
        // valor predeterminado
        public Reserva (int lacapacidad) {
                System.out.println (capacidad);
                capacidad = lacapacidad;
                System.out.println (capacidad);
        }
}

class demovalor {
        static public void main (String []arg) {
                new Reserva (1000);
        }
}
```

visualizará sucesivamente el valor que el núcleo ha dado al atributo capacidad tras la inicialización (2) y posteriormente el valor que le asigna el constructor (1000).

Añadamos que los atributos no inicializados explícitamente por el desarrollador tienen valores predeterminados, iguales a cero. Así, si no se inicializa capacidad:

```
class Reserva {
        int capacidad:
        public Reserva (int lacapacidad) {
                System.out.println (capacidad);
                capacidad = lacapacidad;
                System.out.println (capacidad);
        }
```

```
}
```

el programa indicará los valores 0 seguido de 1000.

EL LENGUAJE JAVA

GRAMATICA

En general la gramática de java se parece a la del C/C++, vamos a ver los puntos más importantes para poder escribir un programa en java.

COMENTARIOS

En Java hay tres tipos de comentarios:

```
// comentarios para una sola línea

/* comentarios de una o
     más líneas
*/

/** comentario de documentación, de una o más líneas
*/
```

Los dos primeros tipos de comentarios son los que todo programador conoce y se utilizan del mismo modo. Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de Java *javadoc*. Dichos comentarios sirven como descripción del elemento declarado permitiendo generar una documentación de nuestras clases escrita al mismo tiempo que se genera el código.

En este tipo de comentario para documentación, se permite la introducción de algunos tokens o palabras clave, que harán que la información que les sigue aparezca de forma diferente al resto en la documentación.

IDENTIFICADORES

Los identificadores nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar.

En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos. **Se distinguen las mayúsculas de las minúsculas** y no hay longitud máxima.

Serían identificadores válidos:

```
identificador
nombre_usuario
Nombre_Usuario
_variable_del_sistema
$transaccion
```

y su uso sería, por ejemplo:

```
int contador_principal;
char _lista_de_ficheros;
float $cantidad_en_Ptas;
```

Palabras clave

Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como indentificadores:

abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while

Palabras reservadas

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico. Son:

- `cast` `future` `generic` `inner`
- `operator` `outer` `rest` `var`

TIPOS DE DATOS

En Java existen 8 tipos primitivos de datos, que no son objetos, aunque el lenguaje cuenta con uno equivalente para cada uno de ellos. Los tipos básicos se presentan en la siguiente tabla:

Tipo	**Tamaño / Formato**	**Descripción**
byte	8-bit complemento a 2	Entero de un byte
short	16-bit complemento a 2	Entero corto
int	32-bit complemento a 2	Entero
long	64-bit complemento a 2	Entero largo
float	32-bit IEEE 754	Punto flotante, precisión simple
double	64-bit IEEE 754	Punto flotante, precisión doble
char	16-bit caracter Unicode	Un carácter
boolean	true, false	Valor booleano (verdadero o falso)

El tamaño de estos tipos está fijado, siendo independiente del microprocesador y del sistema operativo sobre el que esté implementado. Esta característica es esencial para el requisito de la portabilídad entre distintas plataformas.

La razón de que se codifique los char con 2 bytes es para permitir implementar el juego de caracteres Unicode, mucho más universal que ASCII, y sus numerosas extensiones.

Java provee para cada tipo primitivo una clase correspondiente: Boolean, Character, Integer, Long, Float y Double.

¿Por qué existen estos tipos primitivos y no sólo sus objetos equivalentes? La razón es sencilla, por eficiencia. Estos tipos básicos son almacenados en una parte de la memoria conocida como el *Stack,* que es manejada directamente por el procesador a través de un registro apuntador *(stack pointer).* Esta zona de memoria es de rápido acceso, pero tiene la desventaja de que el compilador de java debe conocer, cuando está creando el programa, el tamaño y el tiempo de vida de todos los datos allí almacenados para poder generar código que mueva el *stack pointer,* lo cual limita la flexibilidad de los programas. En cambio, los objetos son creados en otra zona de memoria conocida como *Heap.* Esta zona es de propósito general y, a diferencia del *Stack,* el compilador no necesita conocer ni el tamaño, ni el tiempo de vida de los datos allí alojados. Este enfoque es mucho más flexible pero, en contraposición, el tiempo de acceso a esta zona es más elevado que el necesario para acceder al *stack.*

Aunque en la literatura se comenta que Java eliminó los punteros, esta afirmación es inexacta. Lo que no se permite es la aritmética de punteros. Cuando estamos manejando un objeto en Java, realmente estamos utilizando un *handie* a dicho objeto. Podemos definir un *handie* como una variable que contiene la dirección de memoria donde se encuentra el objeto almacenado.

VARIABLES

En un programa informático, hay que manipular los datos. Estos son accesibles, en general, mediante la utilización de:

parámetros de métodos o de funciones;
variables locales;
variables globales;
atributos de objetos.

En Java las variables globales no existen, el resto, parámetros, variables locales y atributos de objetos, se definen y manipulan de la misma forma que en C++.

Género de las variables

En informática hay dos maneras de almacenar las variables en un método:

- se almacena el valor de la variable;
- se almacena la dirección donde se encuentra la variable.

En Java:

- los tipos elementales se manipulan directamente: se dice que se manipulan por valor;
- los objetos se manipulan a través de su dirección: se dice que se manipulan por referencia.

Por ejemplo:

```
void Metodo() {
      int var1  = 2;
      Objeto var2 = new Objeto();
      var1  = var1  + 3;
      var2.agrega  (3);
}
```

En el ejemplo anterior, var1 representa físicamente el contenido de la variable mientras que var2 sólo representa la dirección que permite acceder a ella: se crea un objeto en algún lugar de la memoria y var2 representa físicamente el medio de acceder a él.

Cuando se añade 3 a var1, se modifica la variable var1, mientras que el método agrega (3) no modifica var2, sino el objeto designado por var2.

A priori preferirá var1, que le parecerá más simple. Es cierto, pero este mecanismo es mucho más restrictivo e impide por ejemplo compartir datos entre diferentes partes de la aplicación.

Además, el almacenamiento al estilo de var2 es indispensable cuando los objetos son algo más que tipos simples.

En Java, para todas las variables de tipo básico se accede al valor asignado a ellas directamente (no se conoce la dirección de memoria que ocupa). Para las demás (arrays, clases o interfaces), se accede a la variable a través de un puntero. El valor del puntero no es accesible ni se puede modificar como en C/C++; Java no lo necesita, además, esto atentaría contra la robustez del lenguaje.

De hecho, en Java no existen los tipos pointer, struct o union. Un objeto es más que una estructura, y las uniones no se hacen necesarias con un método de programación adecuado (además se evita la posibilidad de acceder a los datos incorrectamente).

Asignaciones a variables

Se asigna un valor a una variable mediante el signo =

```
Variable = Constante | Expresión;
```

Por ejemplo:

```
suma = suma + 1;
precio = 1.05 * precio;
```

Inicialización de las variables

En Java no se puede utilizar una variable sin haberla inicializado previamente. El compilador señala un error cuando se utiliza una variable sin haberla inicializado.

Veamos un ejemplo. La compilación de:

```
import java.io.*;
class DemoVariable {
      public static void main (String argv[]) {
            int noInicializado;
            System.out.println("Valor del entero: "+nolnicializado);
      }
}
```

provoca el error siguiente:

```
c: \ProgramasJava\cramatica>javac init.java
init.java:6: Variable noInicializada may not have been initialized
System.out.println ("Valor del entero:"+ nolnicializado);

1 error
```

OPERADORES

Los operadores de Java son muy parecidos en estilo y funcionamiento a los de C. En la siguiente tabla aparecen los operadores que se utilizan en Java, por orden de precedencia:

operadores	
posfijos	[] . (parámetros) expr++ expr--
Operadores unarios	++expr --expr +expr –expr ~ !
Creación y "cast"	new (tipo)
Multiplicativos	* / %
Aditivos	+ -
Desplazamiento	<< >> >>>
Relacionales	< > <= >= instanceof
Igualdad	== !=
AND bit a bit	&
OR exclusivo bit a bit	^
OR inclusivo bit a bit	|
AND lógico	&&
OR lógico	||
Condicional	? :
Asignación	= += -= *= /= % = ^= &= |= <<= >>= >>>=

- Los operadores numéricos se comportan como esperamos. hay operadores unarios y binarios, según actúen sobre un solo argumento o sobre dos.

Operadores unarios

Incluyen, entre otros: +,-, ++, - -, ~, !, (tipo)
Se colocan antes de la constante o expresión (o, en algunos casos, después). Por ejemplo:

```
-cnt;                  //cambia de signo; por ejemplo si cnt es 12 el
                       //resultado es -12 (cnt no cambia)
++cnt;                 //equivalen a cnt+=1;
cnt++;
--cnt;                 //equivalen a cnt-=1;
cnt--;
```

Operadores binarios

Incluyen, entre otros: +, -, *, /, %
Van entre dos constantes o expresiones o combinación de ambas. Por ejemplo:

```
cnt + 2                //devuelve la suma de ambos.
promedio + (valor/2)
horas / hombres;             //división.
acumulado % 3;               //resto de la división entera entre ambos.
```

Nota: + sirve también para concatenar cadenas de caracteres. Cuando se mezclan Strings y valores numéricos, éstos se convierten automáticamente en cadenas:

```
"La frase tiene " + cant + " letras"
```

se convierte en:

```
"La frase tiene 17 letras"        //suponiendo que cant = 17
```

- Los operadores relacionales devuelven un valor booleano.
- El operador = siempre hace copias de objetos, marcando los antiguos para borrarlos, y ya se encargará el garbage collector de devolver al sistema la memoria ocupada por el objeto eliminado.

Veamos algunos ejemplos:

- **[ ]** define arreglos: `int lista [ ];`
- **(params)** es la lista de parámetros cuando se llama a un método: `convertir (valor, base);`
- **new** permite crear una instancia de un objeto: `new contador();`

- **(type)** cambia el tipo de una expresión a otro: `(float) (total % 10);`
- **>>** desplaza bit a bit un valor binario: `base >> 3;`
- **<=** devuelve "true" si un valor es menor o igual que otro: `total <= maximo;`
- **instanceof** devuelve "true" si el objeto es una instancia de la clase: `papa instanceof Comida;`
- || devuelve "true" si cualquiera de las expresiones es verdad: `(a<5) || (a>20).`

Separadores

Sólo hay un par de secuencias con otros caracteres que pueden aparecer en el código Java; son los separadores simples, que van a definir la forma y función del código. Los separadores admitidos en Java son:

() - paréntesis. Para contener listas de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.

{} - llaves. Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.

[ ] - corchetes. Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.

; - punto y coma. Separa sentencias.

, - coma. Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.

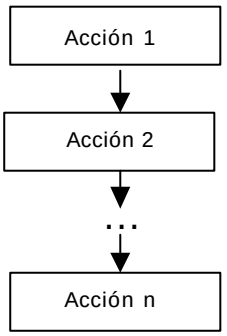
. - punto. Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

ESTRUCTURAS DE CONTROL

Es la manera como se van encadenando, uniendo entre sí las acciones, dando origen a los distintos tipos de estructuras.

Estructura secuencial

La estructura secuencial es aquélla en la que una acción sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.



Acción 1
Acción 2
...
Acción n

Estructuras selectivas (alternativas, de decisión)

Cuando el programa desea especificar dos o más caminos alternativos, se deben utilizar estructuras selectivas o de decisión. Una instrucción de decisión o selección evalúa una condición y en función del resultado de esa condición se bifurcará a un determinado punto.

Las estructura selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

En la estructura selectiva se evalúa una condición y en función del resultado de la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabra en pseudocódigo, con una figura geométrica en forma de rombo.

Las estructuras selectivas o alternativas pueden ser:

- Simples
- Dobles.
- Múltiples.

Alternativa simple

La estructura alternativa simple **si_entonces**, ejecuta una determinada acción cuando se cumple una determinada condición. La selección **si-entonces** evalúa la condición:

- Si la condición es **verdadera,** entonces ejecuta la acción (simple o contar de varias acciones)

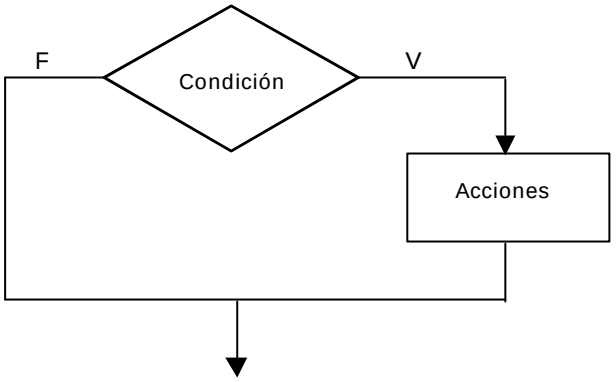
- Si la condición es **falsa,** entonces no hace nada.

La representación gráfica de la estructura condicional simple se ve a continuación.

La estructura alternativa simple **si_entonces**, ejecuta una determinada acción cuando se cumple una determinada condición. La selección **si-entonces** evalúa la condición:

- Si la condición es **verdadera,** entonces ejecuta la acción (simple o contar de varias acciones)
- Si la condición es **falsa,** entonces no hace nada.

La representación gráfica de la estructura condicional simple se ve a continuación.



Si condición **Entonces**

Acciones

Fin_Si

Observe que las palabras del pseudocódigo Si y Fin_Si se alinean verticalmente indentando (sangrando) la acción o bloque de acciones.

Afirmemos esto con un ejemplo en Java: Sea generar dos números al azar, y si el segundo es mayor emitir un mensaje informándolo.

```
public class PruebaMay{
        static int numero1 = (int)(10000*Math.random());
        static int numero2 = (int)(10000*Math.random());
        public static void main(String args[]){
                System.out.println("Primer  numero " + numero1);
                System.out.println("Segundo numero " + numero2);
                if(numero2 > numero1) System.out.println("El segundo numero es el mayor");
        }
};
```

Primer proceso

```
Primer  numero 1337
Segundo numero 7231
El segundo numero es el mayor
Process Exit...
```

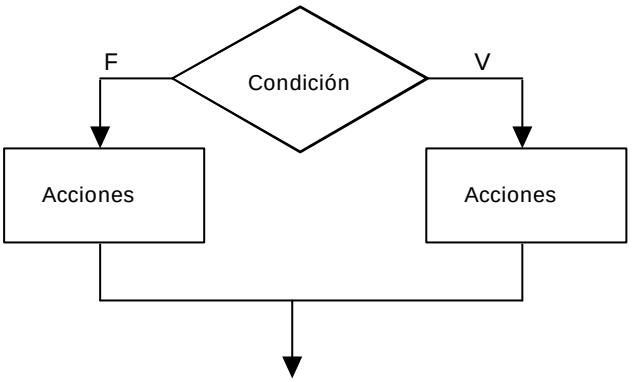
Segundo

```
Primer  numero 5410
Segundo numero 929
Process Exit...
```

Alternativa doble

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición.

Si la condición es verdadera, se ejecuta la acciónes_1 y si es falsa, se ejecuta la acciónes_2.



Si Condición **Entonces**
Acciones
Sino
Acciones
Fin_Si

Observe que en el pseudocódigo las acciones que dependen de entonces y sino están indentadas en relación con las palabras Si y Fin_Si; este procedimiento aumenta la legibilidad de la estructura y es el medio idóneo para representar algoritmos.

Ejemplo Java: Generar dos números al azar indicando cual de ellos es el mayor.

```
public class PruebaMay{
	static int numero1 = (int)(10000*Math.random());
	static int numero2 = (int)(10000*Math.random());
	public static void main(String args[]){
		System.out.println("Primer  numero " + numero1);
		System.out.println("Segundo numero " + numero2);
		if(numero2 > numero1) System.out.println("El segundo numero es el mayor");
		else System.out.println("El primer numero es el mayor");
	}
};
```

Un par de procesos:

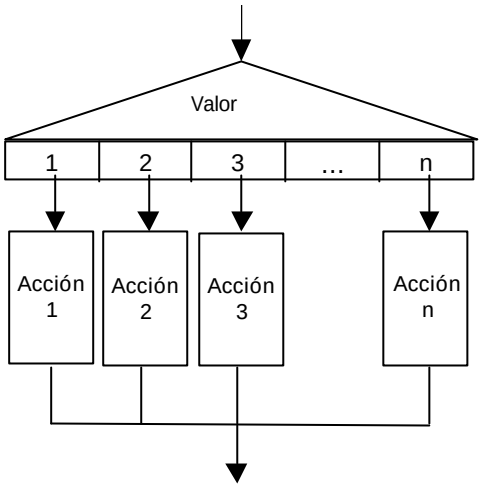
```
Primer  numero 5868
Segundo numero 654
El primer numero es el mayor
Process Exit...

Primer  numero 3965
Segundo numero 7360
El segundo numero es el mayor
Process Exit...
```

Alternativa múltiple

Con frecuencia, es necesario que existan más de dos elecciones posibles. Por ejemplo: en la resolución de la ecuación de segundo grado existen tres posibles alternativas o caminos a seguir, según que el discriminante sea negativo, nulo o positivo. Este problema, se podría resolver por estructuras alternativas simples o dobles, anidadas o en cascada; sin embargo, con este método, si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos: 1, 2, 3,..., n. Según que elija uno de estos valores en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá sin determinado camino entre los n posibles.



Según Valor **Hacer**

Valor 1: Acción 1

Valor 2: Acción 2

Valor 3: Acción 3

...

Valor n: Acción n

Fin Según

En Java, su implementación requiere de las siguientes palabras reservadas: **switch case.. .break. . .default**

```
switch (expresión_entera) {
	case (valor1) : instrucciones_1; [break;]
	case (valor2) : instrucciones_2; [break;]
	...
	case (valorN) : instrucciones_N; [break;]
	default: instrucciones_por_defecto;
}
```

Ejemplo: Expresar **literalmente** el resto de la división entera por 6 de un numero generado al azar.

```
	public class Switch1{
		static int numero = (int)(1000*Math.random());
		static int resto  = numero%6;
		public static void main(String args[]){
			System.out.println("El numero random: " + numero);
			System.out.println("Su resto en la division: " + resto);
			switch (resto) {
				case (0):
					System.out.println("Confirmo, el resto es cero");
					break;
				case (1):
					System.out.println("Confirmo, el resto es uno");
					break;
				case (2):
					System.out.println("Confirmo, el resto es dos");
					break;
				case (3):
					System.out.println("Confirmo, el resto es tres");
					break;
				default:
					System.out.println("El resto es mayor que tres");
					break;
			}
		}
	};
```

Ejecutamos:

```
El numero random: 326
Su resto en la division: 2
Confirmo, el resto es dos
Process Exit...
```

Si sacamos los “breaks” la alternativa múltiple es también acumulativa, ejemplo:

```
public class Switch2{
      static int numero = (int)(5*Math.random());
      public static void main(String args[]){
            System.out.println("El numero random: " + numero);
            switch (numero){
                  case (5):
                        System.out.println("Pasando por case (5)");
                  case (4):
                        System.out.println("Pasando por case (4)");
                  case (3):
                        System.out.println("Pasando por case (3)");
                  case (2):
                        System.out.println("Pasando por case (2)");
                  default:
                        System.out.println("Pasando por default");
            }
      }
};
```

y nos da:

```
El numero random: 4
Pasando por case (4)
Pasando por case (3)
Pasando por case (2)
Pasando por default
Process Exit...
```