Unidad I

Algoritmos con TIPOS DE DATOS SIMPLES

Año 2008

" Ojalá fuera yo mas prudente ¡Ojalá fuera tan astuto como la serpiente! Y si un dia la prudencia me abandona que mi orgullo vuele junto con mi locura. Así comenzó el ocaso de Zarathustra. "

Federico Nietzsche

Autor: Ing. Tymoschuk, Jorge
Colaboración: Ing. Guzmán, Analía

Indice (Unidad I - Algoritmos con tipos de datos simples)

Objetivos de la Unidad 3
Pasos a seguir en la resolución de un problema . . 3
Introducción a la POO 4
Que es la programación Orientada a Objetos 5
Componentes básicos de la POO . . 5
Características de la POO . . . 5
Programación estructurada, Programación orientada a objetos 9
Lenguaje Java, características . . . 11
Instalación de java . . . 13
Compilación y ejecución de un programa 13
Gramática del lenguaje Java 14
Comentarios 14
Identificadores, palabras clave y reservadas 15
Concepto de Dato 15
Concepto de Información, dif. entre datos e información 16
Tipos de Datos Simples, Variables 17
Genero de las variables, asignación, inicialización 18
Operadores: unarios, binarios . . . 19
Separadores 21
Estructuras de control: secuencial, selectivas 21
Alternativa simple, doble 21
Alternativa múltiple . . . 23
Estructuras de Control: Bucles . . . 25
El Bucle While 25
Terminaciones anormales de un ciclo . 26
Bucles controlados por centinela . . 27
Diseño eficiente de bucles . . 27
Bucles controlados por banderas . . 29
Sentencias de quiebre de control . 30
Repetición: El Bucle For . . . 31
Repetición: El Bucle do-while . . 34
Bucles Anidados 35
Ejercicios propuestos para el alumno . . 36

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.

Objetivos de la Unidad

Desarrollar la capacidad de razonamiento y lógica necesarios para identificar problemas algorítmicos que usan tipos de datos simples y resolverlos. Esto implica: abordar problemas reales, analizarlos, definir la estrategia de su resolución, explicitar los algoritmos necesarios y codificarlos en un lenguaje de programación. Normalmente esto implica seguir los siguientes

Pasos a seguir en la resolución de un problema

- **Interpretación 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.

- **Abstracción** 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.

- Formular la estrategia de resolución (Diagrama o pseudocódigo) 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).

- Verificar dicha estrategia (PRUEBA DE ESCRITORIO)

- Codificar en un lenguaje de programación.

- Depurar errores de codificación.

- Pruebas, correcciones,...

Interpretación del problema (Análisis)

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. **Interpretación del problema (Análisis)**
   En este caso tan sencillo es evidente que el dato de entrada será el radio y los resultados la superficie y el perímetro.

2. **Abstracción**
   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.

La mejor forma de aprender es haciendo, resolviendo casos concretos. Pero hay un mínimo de orientación en el uso de las herramientas que debemos conocer. Por ello comenzamos con una

Introducción a la POO

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 ellos tiene ciertas características y se comporta de una manera determinada. Si los 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 mismas. 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 Programación 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:

**0 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.

**1 Elementos de interfaces gráficos de usuarios*

Ventanas.
Objetos gráficos (líneas, rectángulos, círculos).
Menús.

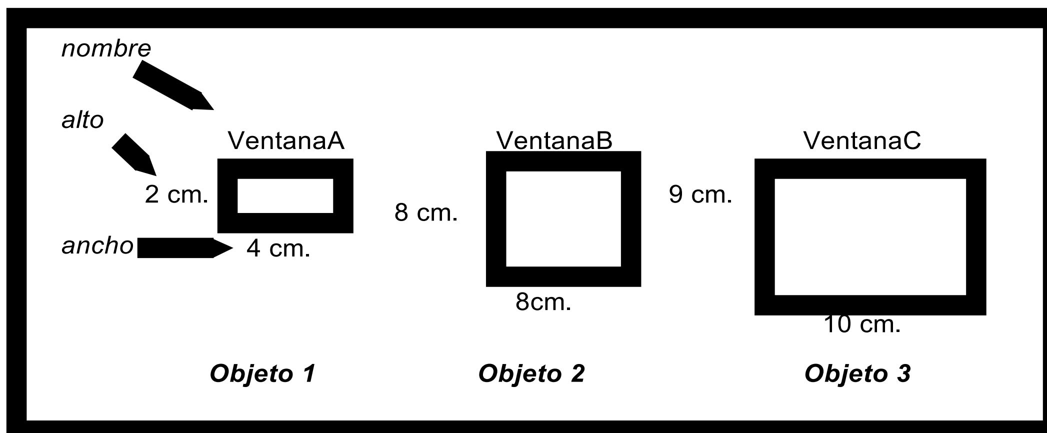
**2 Estructuras de datos*

Vectores.
Listas.
Arboles binarios.

**3 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).

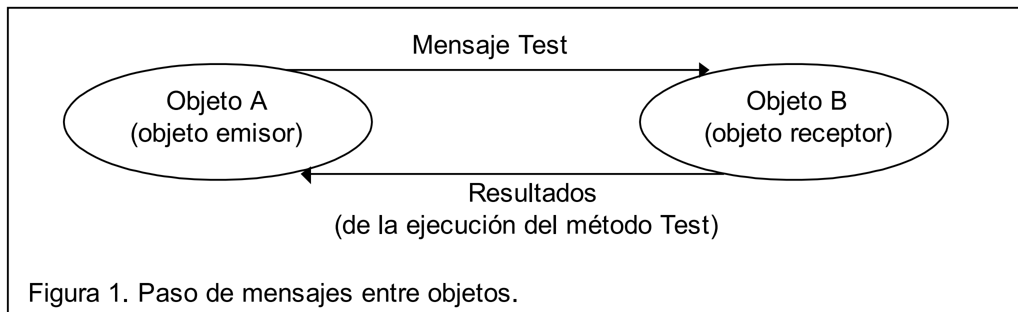


Figura 1. Paso de mensajes entre objetos.

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.

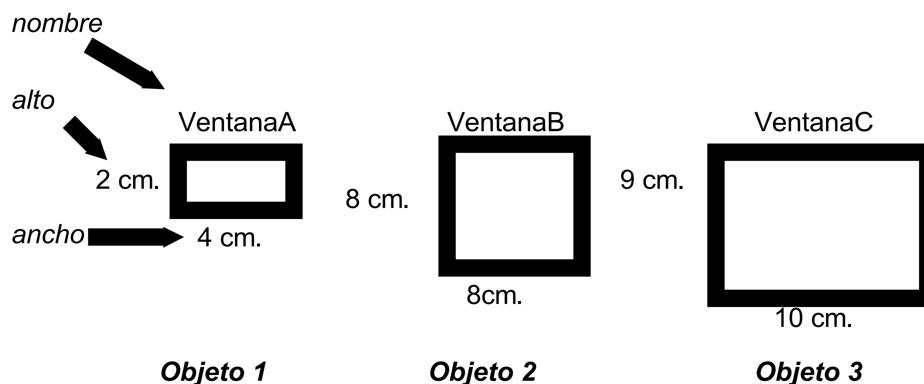
Una clase puede tener muchas instancias y cada una es un objeto independiente.

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



Características 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.

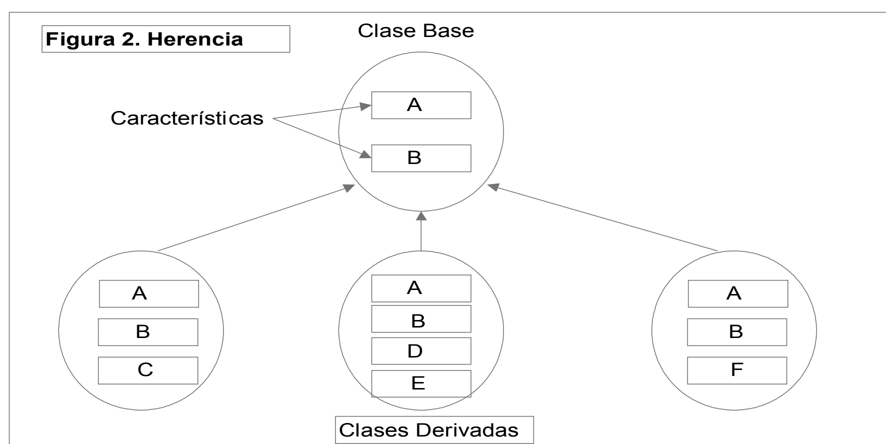
Herencia La herencia es la propiedad que permite a los objetos construirse a partir de otros objetos. El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases. Así por ejemplo, las clases de animales se dividen en mamíferos, anfibios, insectos, pájaros, 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 siguiente. 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 al mensaje **imprimirListado()** , claro que las acciones que se ejecuten(respuesta) **monitor. imprimirListado()** son distintas a las de **archivo. imprimirListado()**

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; Si usaremos lenguajes de la llamada Vertiente Imperativa(El programador detalla minuciosamente las acciones que la computadora debe realizar) disponemos de dos paradigmas: Programación Estructurada y Orientada 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 el segundo es una evolución del primero, es el que nosotros vamos a usar ya que es la actual realidad del mercado, es la que mejor rendimiento tiene, se está usando desde hace bastantes años y es la que tiene mayor soporte en cuanto a tecnologías de análisis, diseño y desarrollo de software.

A continuación una breve comparación de estos dos paradigmas. Nos servirá para entender mejor como tendremos que resolver los problemas en el paradigma que adoptamos.

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 inicialmente dijimos, aprovecha la capacidad humana de conocer **conceptos.** Los humanos tenemos conocimiento concreto de conceptos generales, esto nos posibilitan interactuar con los demás. Quienes sean versados o especialistas en determinadas disciplinas, conocen conceptos específicos, no tan comunes. Pueden interactuar fácil y eficientemente con sus pares, no tanto con quienes no conocen, o tienen una idea vaga del significado de esos conceptos específicos.

En POO, el **concepto se implementa en clases** (*class*). **La clase es la unidad básica de programación.** Es una unidad que contiene los atributos y todos los métodos necesarios a su tratamiento.

Entonces **cada clase contiene datos junto con un conjunto de métodos** 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 o métodos.***

Una instancia, cada caso particular de una clase es un **objeto.**

El foco de atención está sobre los objetos, en lugar de estarlo sobre las funciones. Son los **sustantivos** que 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:

Enunciado

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

Interpretación del problema (Análisis)

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

Abstracción

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

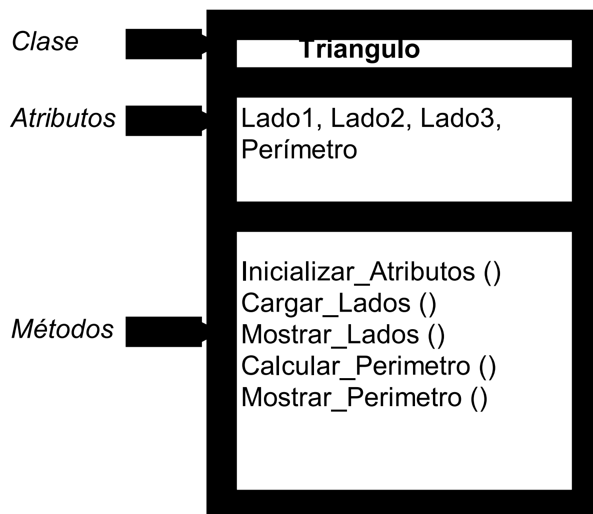
Identificar sus atributos: podemos decir que todo triángulo tiene como datos esenciales los tres lados, además, nuestro problema en particular necesitará el valor del perímetro, que es un datos calculable a partir de los tres lados y que puede figurar como atributo o no dependiendo si se quiere guardar 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

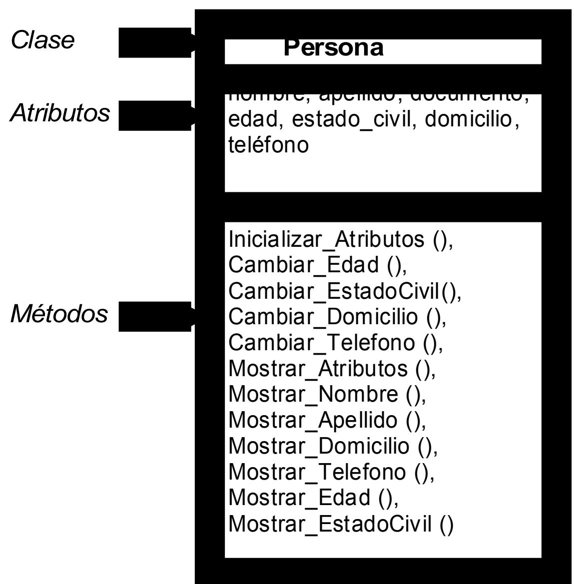


Ejemplos: dadas las siguientes entidades definirlas, identificar la clase, los atributos, los métodos y los mensajes correspondientes.

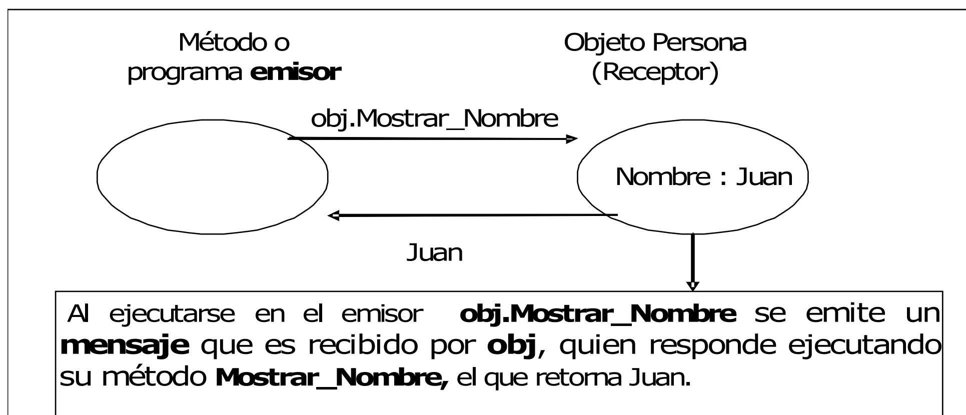
1. Persona 2. Cliente 3. Proveedor 4. Producto
5. Computadora 6. Mesa 7. Automóvil 8. Avión
9. Cuenta bancaria 10. Comprobante de factura
11. Definir cualquier entidad de la vida real, que se utilice en la facultad, trabajo, hogar, etc.

Resoluciones

1.1. Persona



Ej : 1 Procesamiento de un mensaje.



Lo que hasta ahora hemos visto es una introducción, un posicionamiento en lo que esta asignatura debe trabajar; pero es poco lo que podemos hacer **sin herramientas**, es hora de comenzar a conocerlas.

Suponemos que UD tiene acceso a una computadora, la necesitará. Y en ella, además del software de uso habitual necesitaremos del lenguaje Java, que usaremos para codificar nuestros programas, y de un entorno de desarrollo, un IDE (Integrated Development Environment. Hablemos de ello.

Lenguaje Java, características

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.

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**(Graphic User Interface): la caja de herramientas para la creación de interfaces gráficas de usuario de Java simplifica y facilita la escritura de programas GUI, 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.

Instalación de Java

Para configurar su ambiente Java, simplemente baje la versión mas reciente de JDK (java development Kid) del sitio Web de Sun Microsystems http://java.sun.com/

Clic downloads, HOY 12/2007, puede bajar :

- JDK 6 Update 3 with Java EE 5 SDK Update 3 (Sirve también para la asignatura Paradigmas, luego debe optar por un IDE y bajarlo)

- JDK 6 Update 3 with NetBeans IDE 6

Otra opción es concurrir al Laboratorio de Sistemas, lleve un CD. Le hacen una copia de un día para otro

Para instalación, siga las instrucciones del wizard (Ayudante)

Compilación y ejecución de un programa

Normalmente Ud usará un IDE (Integrated Develpment Environment, Entorno de Desarrollo Integrado). Si bien es posible codificar, compilar, ejecutar programas fuera de un IDE (Y los libros lo enseñan) nadie hace esto. Algún IDE, hay muchos (JavaCreador, Blue J, NetBeans IDE, Eclipse, etc, etc) es lo que Ud va usar. En el Laboratorio, con una cartilla, Ud aprenderá a usarlo. Dependiendo del equipo que disponga, con máquinas "humildes" podrá usar los dos primeros y con una computadora actual podrá usar NetBeans, muy recomendable, con precompilación, ayuda en línea, etc.

Veamos un programa Java, muy simple

```
// programa en Java, grabado en un archivo llamado "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");
            Print.prt(sl);
}
```

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 será la que se ejecuta cuando en nuestro IDE seleccionamos **Run, Run main proyect.**

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.

Los programas Java se organizan en proyectos constituidos por una o varias clases. Cuando vamos a trabajar en un proyecto necesitamos abrirlo previamente, o crearlo si es nuevo. Las clases se graban en archivos .Java. Un archivo .java, conteniendo código origen (source) se puede:

- **editar**, el programador codifica una clase en Java. Puede ser una nueva, modificar uno existente. Si existente, usando el IDE posicionamos cursor sobre el .java, abrir. Si nueva, new, archivo java vacio (Por ejemplo)
- **compilar**, El compilador de Java traduce el programa Java a códigos de bytes, que es el lenguaje que entiende la JVM (Java Virtual Machina, Máquina Virtual Java)el intérprete de Java. Si el programa se compila correctamente, se producirá un archivo llamado Hello.class. Éste es el archivo que contiene los códigos de bytes que serán interpretados durante la fase de ejecución.
- **Cargar**, 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. Esto ocurre cuando UD use la opción **run** de su entorno de desarrollo. Llamamos al programa Hello una aplicación. Las aplicaciones son programas que son ejecutados por la Máquina virtual de Java. El cargador de clases también se ejecuta cuando un applet de Java(Veremos esto en PPR) se carga en un navegador de la World Wide Web, como Navigator de Netscape o Explorer de Microsoft.
- **Verificar**, antes de que JVM pueda ejecutar los códigos de bytes, éstos son verificados por el verificador de códigos de bytes. Esto asegura que los códigos de bytes son válidos y que no violan las restricciones de seguridad de Java. Java cumple 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.
- **Ejecutar**, por último 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 si las correcciones surtieron el efecto que se esperaba.

Gramática del lenguaje Java

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

Concepto de Datos

Dato: Antecedente que permite llegar mas fácilmente al conocimiento de una cosa. **Información:** Noción susceptible de ser estudiada por un ordenador.

Definiciones del diccionario PEQUEÑO LAROUSSE ILUSTRADO

Concepto de Datos: Representación en un lenguaje preciso, formalizado de algunos hechos o conceptos, a menudo valores numéricos o alfabéticos, de forma que puedan manipularse por un método de cálculo.

Tomado del libro ALGORITMOS FUNDAMENTALES, Donald E, KNUTH

A continuación transcribimos parcialmente un trabajo publicado por Sergio D'Ambrosio, dambrosio@cantv.net, I.U.P. "Santiago Mariño" Puerto Ordaz, Ing. De Sistemas en www.monografias.com.

Datos

Indice

1. El Concepto de Datos
2. El Concepto de Información

3. Diferencia entre Datos e información

1. El Concepto de Datos

Datos son los hechos que describen sucesos y entidades."Datos" es una palabra en plural que se refiere a más de un hecho. A un hecho simple se le denomina "data-ítem" o elemento de dato.
Los datos son comunicados por varios tipos de símbolos tales como las letras del alfabeto, números, movimientos de labios, puntos y rayas, señales con la mano, dibujos, etc. Estos símbolos se pueden ordenar y reordenar de forma utilizable y se les denomina información.
Los datos son símbolos que describen condiciones, hechos, situaciones o valores. Los datos se caracterizan por no contener ninguna información. Un dato puede significar un número, una letra, un signo ortográfico o cualquier símbolo que represente una cantidad, una medida, una palabra o una descripción.
La importancia de los datos está en su capacidad de asociarse dentro de un contexto para convertirse en información. Por si mismos los datos no tienen capacidad de comunicar un significado y por tanto no pueden afectar el comportamiento de quien los recibe. Para ser útiles, los datos deben convertirse en información para ofrecer un significado, conocimiento, ideas o conclusiones.

2. El Concepto de Información

La información no es un dato conjunto cualquiera de ellos. Es más bien una colección de hechos significativos y pertinentes, para el organismo u organización que los percibe. La definición de información es la siguiente: Información es un conjunto de datos significativos y pertinentes que describan sucesos o entidades.
DATOS SIGNIFICATIVOS. Para ser significativos, los datos deben constar de símbolos reconocibles, estar completos y expresar una idea no ambigua.
Los símbolos de los datos son reconocibles cuando pueden ser correctamente interpretados. Muchos tipos diferentes de símbolos comprensibles se usan para transmitir datos.
La integridad significa que todos los datos requeridos para responder a una pregunta específica están disponibles. Por ejemplo, un marcador de béisbol debe incluir el tanteo de ambos equipos. Si se oye el tanteo "New York 6" y no oyes el del oponente, el anuncio será incompleto y sin sentido.
Los datos son inequívocos cuando el contexto es claro. Por ejemplo, el grupo de signos 2-x puede parecer "la cantidad 2 menos la cantidad desconocida llamada x" para un estudiante de álgebra, pero puede significar "2 barra x" a un vaquero que marca ganado. Tenemos que conocer el contexto de estos símbolos antes de poder conocer su significado.
Otro ejemplo de la necesidad del contexto es el uso de términos especiales en diferentes campos especializados, tales como la contabilidad. Los contables utilizan muchos términos de forma diferente al público en general, y una parte de un aprendizaje de contabilidad es aprender el lenguaje de contabilidad. Así los términos Debe y Haber pueden significar para un contable no más que "derecha" e "izquierda" en una contabilidad en T, pero pueden sugerir muchos tipos de ideas diferentes a los no contables.
DATOS PERTINENTES. Decimos que tenemos datos pertinentes (relevantes) cuando pueden ser utilizados para responder a preguntas propuestas.
Disponemos de un considerable número de hechos en nuestro entorno. Solo los hechos relacionados con las necesidades de información son pertinentes. Así la organización selecciona hechos entre sucesos y entidades particulares para satisfacer sus necesidades de información.

3. Diferencia entre Datos e información

1. Los Datos a diferencia de la información son utilizados como diversos métodos para comprimir la información a fin de permitir una transmisión o almacenamiento más eficaces.
2. Aunque para el procesador de la computadora hace una distinción vital entre la información entre los programas y los datos, la memoria y muchas otras partes de la computadora no lo hace. Ambos son registradas temporalmente según la instrucción que se le de. Es como un pedazo de papel no sabe ni le importa lo que se le escriba: un poema de amor, las cuentas del banco o instrucciones para un amigo. Es lo mismo que la memoria de la computadora. Sólo el procesador reconoce

la diferencia entre datos e información de cualquier programa. Para la memoria de la computadora, y también para los dispositivos de entrada y salida (E/S) y almacenamiento en disco, un programa es solamente más datos, más información que debe ser almacenada, movida o manipulada.
3. La cantidad de información de un mensaje puede ser entendida como el número de símbolos posibles que representan el mensaje."los símbolos que representan el mensaje no son más que datos significativos.
4. En su concepto más elemental, la información es un mensaje con un contenido determinado emitido por una persona hacia otra y, como tal, representa un papel primordial en el proceso de la comunicación, a la vez que posee una evidente función social. A diferencia de los datos, la información tiene significado para quien la recibe, por eso, los seres humanos siempre han tenido la necesidad de cambiar entre sí información que luego transforman en acciones. "La información es, entonces, conocimientos basados en los datos a los cuales, mediante un procesamiento, se les ha dado significado, propósito y utilidad"

Fin del trabajo de Sergio D'Ambrosio

EL CONCEPTO SINTÁCTICO O FORMAL DE CANTIDAD DE INFORMACIÓN

Veamos todavía con mayor detalle el concepto de cantidad de información: "***La cantidad de información de un sistema, compuesto de partes,*** *se define entonces a partir de las probabilidades que pueden asignarse a cada uno de sus componentes, en un conjunto de sistemas que se suponen estadísticamente homogéneos los unos con los otros ; o también a partir del* ***conjunto de las combinaciones que es posible realizar con sus_componentes,*** *lo que constituye el conjunto de los estados posibles del sistema*". (Los destacados son nuestros.) (Henry Atlan, 1990, p.48.)

Si suponemos que un sistema complejo no es el resultado del mero azar, entonces la ***cantidad de información*** mide el grado de ***improbabilidad*** de que el sistema considerado se haya constituido exclusivamente por azar.(Henry Atlan, 1990, p.49.) De todas las combinaciones posibles entre los elementos constitutivos del sistema, solamente un grupo muy reducido de ellas y, en rigor, una única combinación de tales elementos es la que concretamente realiza el sistema que sea el caso. (Es como armar las piezas de un rompecabezas.)

Y mientras mayor sea el número de elementos que constituyen tal sistema, será más grande la cantidad de información, ya que será todavía más improbable que esos elementos se hayan articulado por simple azar. (Como si se tratara de un rompecabezas con mayor número de piezas.)

Sigue … Tomado de www.homeoint.org/books3/diluciones/sintactico.htm

Tipos de Datos Simples

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 *handle* 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)

<table>
<tr><td>Multiplicativos</td><td>* / %</td></tr>
<tr><td>Aditivos</td><td>+ -</td></tr>
<tr><td>Desplazamiento</td><td><< >> >>></td></tr>
<tr><td>Relacionales</td><td>< > <= >= instanceof</td></tr>
<tr><td>Igualdad</td><td>== !=</td></tr>
<tr><td>AND bit a bit</td><td>&</td></tr>
<tr><td>OR exclusivo bit a bit</td><td>^</td></tr>
<tr><td>OR inclusivo bit a bit</td><td>|</td></tr>
<tr><td>AND lógico</td><td>&&</td></tr>
<tr><td>OR lógico</td><td>||</td></tr>
<tr><td>Condicional</td><td>? :</td></tr>
<tr><td>Asignación</td><td>= += -= *= /= % =
^= &= |= <<= >>= >>>=</td></tr>
</table>

- 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).
- && devuelve "true" si ambas expresiones son 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.

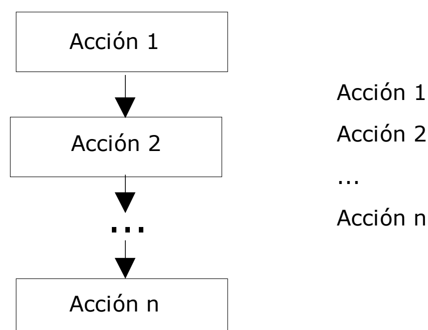
. - punto. Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

Estructuras de Control: Selección

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.



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

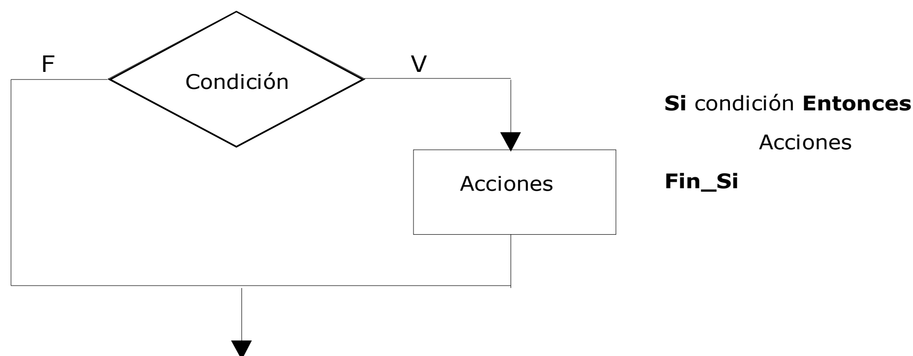
La estructura alternativa simple **si_entonces**, ejecuta una determinada acción cuando se cumple una determinada condición. La selección **si-entonces** evalúa la condición:

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

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



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");
      }
};
```

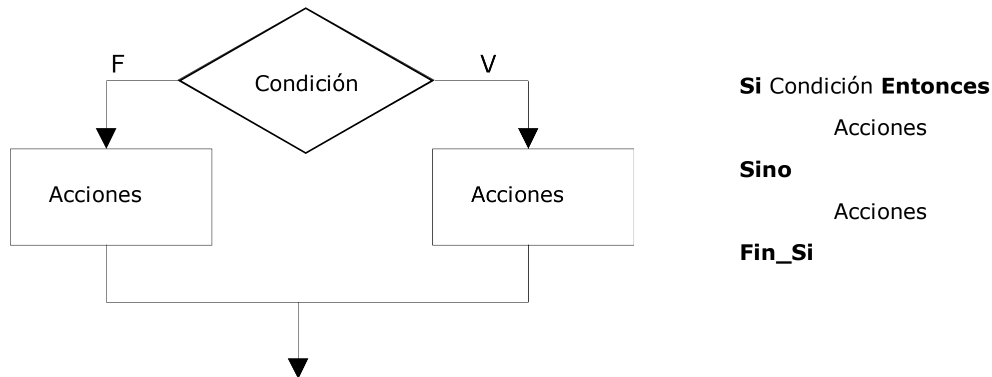
```
Primer proceso

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");
      }
};
```

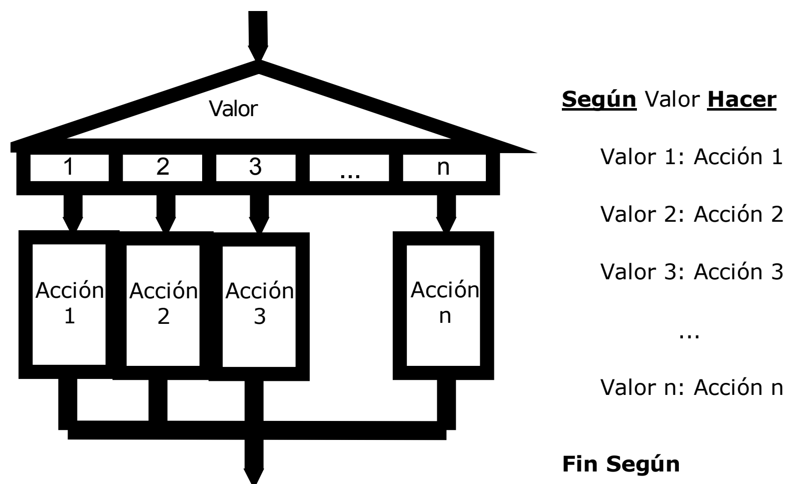
Un par de procesos:

```
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.



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...
```

Estructuras de Control: Bucles

Las computadoras pueden ejecutar una tarea muchas (repetidas) veces con gran velocidad, precisión y fiabilidad. Algo que los humanos encontramos difícil y tedioso. Para ello necesitan de los bucles, o sea estructuras de control iterativas o repetitivas que realizan la repetición o iteración de acciones. Java, como muchos otros lenguajes, soporta tres tipos de estructuras de control: los bucles while, for y do-while. Estas estructuras controlan el número de veces que una acción (sentencia simple) o grupo de acciones (sentencia compuesta) se ejecutan.

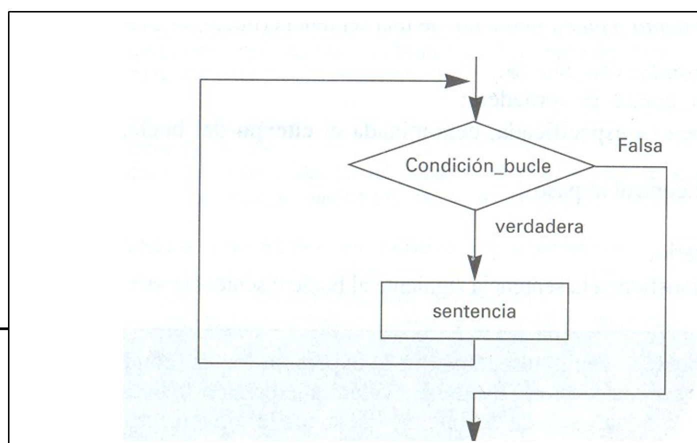
El Bucle While

Un bucle es cualquier construcción de programa que controla la repetición de una sentencia simple o compuesta, denominada cuerpo del bucle. Cada repetición es una iteración del bucle.

Las dos principales cuestiones de diseño en la construcción del bucle son:

- ¿Cuál es el cuerpo del bucle?
- Cuántas veces se iterará?

Un bucle while (Mientras) tiene una condición del bucle (una expresión lógica) que controla la secuencia de repetición.



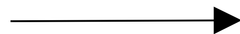
La posición de esta condición
del bucle es anterior al cuerpo,
su modalidad es pretest: se
evalúa la condición, si esta
es verdadera se ejecuta el
cuerpo del bucle.

El diagrama indica que la ejecución de la sentencia o sentencias expresadas se repite mientras la condición del bucle permanece verdadera y termina cuando se hace falsa. También indica el diagrama anterior que la condición del bucle se evalúa antes de que se ejecute el cuerpo del bucle y por consiguiente si esta condición es inicialmente falsa, el cuerpo del bucle no se ejecutará. En definitiva, el cuerpo de un bucle while se ejecutará cero o más veces y luego el control se transfiere a la sentencia siguiente al cuerpo del bucle.

Ejemplo. Visualizar n asteriscos.

```
package while01;

// Este programa ejecuta un bucle while imprimiendo entre 0 y 9 asteriscos

public class Main{
     public static void main(String[] args){
        int cont = (int)(10*Math.random());
        System.out.println("Imprimimos "+cont+" asteriscos");
        while(cont>0){
            System.out.print("*");
            cont--;
        }
        System.out.println("\nDemo terminado !!!");
    }
}
```

El "output" (Salida)



```
run:
Imprimimos 6 asteriscos
******
Demo terminado !!!
```

La variable que representa la condición del bucle se denomina también variable de control del bucle debido a que su valor determina si el cuerpo del bucle se repite. La variable de control del bucle debe ser:
1. Inicializada. Cont se establece a un valor inicial.
2. Comprobada. Se comprueba el valor de cont antes de la iteración.
3. Actualizada. Contador se actualiza (cnt--) dentro del cuerpo del bucle.

Si la variable de control no actualiza el bucle, éste se ejecutará «siempre». Tal bucle se denomina **bucle infinito**. En otras palabras, un bucle infinito (sin terminación) se producirá cuando la condición del bucle permanece verdadera y no se hace falsa en ninguna iteración.

Terminaciones anormales de un ciclo

Un error típico en el diseño de una sentencia while se produce cuando el bucle sólo tiene una sentencia en lugar de varias sentencias como se planeó.
El código siguiente:

```
contador = 1;
while (contador < 25)
      System.out.println("contador: "+contador);
      contador++;
```

visualizará infinitas veces el valor 1. Es decir, entra en un bucle infinito del que nunca sale porque no se actualiza (modifica) la variable de control contador.
La razón es que el punto y coma al final de la línea System.out… hace que el cuerpo del bucle termine allí, aunque aparentemente el sangrado pueda dar la sensación de que el cuerpo de while contiene dos sentencias, incluyendo contador++;

```
// la solución es muy sencilla.
contador = 1;
while (contador < 25){
      System.out.println("contador: "+contador);
      contador++;
      }
```

Bucles controlados por centinela

Frecuentemente, no se conoce con exactitud cuántos elementos de datos se procesarán, incluso porque esto puede depender de los mismos datos.

Un medio para manejar esta situación es instruir al usuario a introducir un único dato definido y especificado denominado centinela como fin de datos. El bucle termina cuando se lee el valor centinela.

```
package while02;
// cuenta/acumula secuencia de números, fin numero = 999,
public class Main {
    public static void main(String[] args){
        int orden = 0; int acumulado = 0; int numero = 0;
        System.out.println("Primer numero, por favor ");
        while((numero = In.readInt()) != 999){
            orden++;
            acumulado+=numero;
            System.out.println("van "+orden+" numeros");
            System.out.println("Un numero, por favor ");
        }
        System.out.println("El acumulado es "+acumulado);
        System.out.println("Demo terminado!!!");
    }
}
```

```
Primer numero, por favor
10
van 1 numeros
Un numero, por favor
20
van 2 numeros
Un numero, por favor
30
van 3 numeros
Un numero, por favor
999
El acumulado es 60
Demo terminado!!!
```

Diseño eficiente de bucles

Una cosa es analizar la operación de un bucle y otra diseñarlo eficientemente.
Hubo un profesor que se tomó el trabajo de evaluar lo que hacían sus alumnos.
Acababa de explicarles el esquema while (Mientras) y les pidió que, usando pseudo código, lo aplicaran a un enunciado. He aquí los resultados:

Sea el siguiente enunciado:

Escribir un algoritmo que permita pelar un número suficiente de patatas para una determinada cantidad de comensales. Por comodidad, las patatas a pelar se disponen en un cesto. Debe tenerse en cuenta que el cesto puede vaciarse en un momento dado.

A continuación diversas soluciones de alumnos como respuesta a este ejercicio, junto con la respectiva evaluación del

Solución 1

```
mientras cesto_no_vacío hacer
      pelar 1 patata;
      pelar 1 patata;
fmientras;
```

Si inicialmente el cesto esta vacío, no se pela ninguna patata.
Si el cesto no esta vacío, se presentan dos casos:
- el número de patatas contenidas en el cesto es par: se pelan todas;
- el número de patatas contenidas en el cesto es impar: el algoritmo es incorrecto puesto que es imposible pelar la segunda.
Nada asegura que el número de patatas peladas sea suficiente, insuficiente o excesivo.

Solución 2

```
mientras cesto_no_vacío hacer
      mientras num_patatas_insuficiente hacer
            pelar 1 patata;
      fmientras;
fmientras;
```

. Si inicialmente el cesto está vacío, no se pela ninguna patata.
. Si el número de patatas resulta suficiente antes de que se vac1e el cesto, no salimos nunca del primer ciclo.
Si el cesto se vacía antes de que el número de patatas sea suficiente, el algoritmo es incorrecto puesto que la acción de pelar es entonces irrealizable. El número de patatas no es nunca suficiente al inicio del algoritmo (salvo que se desee pelar O patatas).

```
fmientras;
```

Correcta, pero no siempre proporciona un numero suficiente de patatas. Si el cesto tiene pocas …

Solución 4

```
mientras num_de_patatas_insuficiente
      si cesto_no_vacio
            Entonces pelar una patata
      sino llenarlo
      fsi
fmientras;
```

Correcta, pero la única acción dentro del bucle consiste unas veces en pelar una patata y otras en llenar el cesto. Hay mejores soluciones…

Solución 5

```
mientras num_de_patatas_insuficiente
      mientras cesto_no_vacio
            pelar 1 patata
      fmientras
fmientras;
```

Si el cesto se vacía antes de que antes que el número de patatas sea suficiente, el algoritmo entra en un bucle indefinido.(el mientras externo se mantiene verdadero, el mientras interno es siempre falso, y no hacemos nada…)
En caso contrario, se pelan todas las patatas del cesto, tanto si el número de peladas es igual o superior a la cantidad deseada

Solución 6

```
mientras num_de_patatas_insuficiente
      si cesto_no_vacio pelar 1 patata
      fsi
fmientras;
```

Si el número de patatas contenido en el cesto es inferior a la cantidad deseada, el algoritmo entra en un bucle indefinido. (el mientras se mantiene verdadero, una vez que el si interno se torna falso, no hacemos mas nada…)

Solución 7

```
mientras cesto_lleno
      si num_de_patatas_insuficiente
            pelar 1 patata
      sino llenar el cesto;
            pelar 1 patata
      fsi
fmientras;
```

Que significa cesto_lleno? (suponemos que es cuando no se pueda agregar ninguna mas).
Si inicialmente no tenemos cesto_lleno, no se pela ***ninguna patata.***
Supongamos que inicialmente tenemos cesto_lleno.
- *en cualquier caso pelamos una patata.*
- *El cesto deja de ser cesto_lleno*

Nos vamos, habiendo pelado ***1 patata***

Solución 8

```
mientras cesto_lleno
      si num_de_patatas_insuficiente
            pelar 1 patata
      fsi
fmientras;
```

Estamos en el mismo desastre que en la anterior solución

Solución 9

```
si cesto_no_vacio
      mientras num_de_patatas_insuficiente
            pelar 1 patata
      fmientras
sino llenar cesto
fsi;
```

Si el número de patatas que necesitamos es mayor que las que contiene inicialmente el cesto, este algoritmo nos obliga a pelar batatas inexistentes…

Solución 10

```
mientras (num_de_patatas_insuficiente o cesto_no_vacio)
      pelar 1 patata
fmientras;
```

Aquí pelaremos patatas por cualquiera de las dos condiciones, pues con 1 verdadera seguiremos pelando.
Entonces, siempre vaciaremos el cesto.
Luego, si aún num_de_patatas_insuficiente, seguiremos pelando patatas inexistentes …

Solución 11

```
mientras num_de_patatas_insuficiente
      si cesto_lleno pelar 1 patata
            sino llenar el cesto;
            pelar 1 patata;
      fsi
fmientras;
```

Proporciona el número de patatas suficiente y se detiene. Claro que tiene una forma de trabajar no demasiado brillante, eso de ir reponiendo cada patata que se pela no es muy práctico…

Solución 12

```
mientras num_de_patatas_insuficiente
      si cesto_vacio llenar el cesto fsi;
      pelar 1 patata;
fmientras;
```

Correctísima. *Y que sencillo que era el algoritmo. En general, si el algoritmo se torna complicado en algo que intuimos, parece simple hay que desconfiar…*

Bucles controlados por banderas

En general, los indicadores o banderas son variables que registran la ocurrencia o no de un determinado evento. Para este uso, es natural el uso de variables tipo boolean. Muy frecuentemente usaremos estas banderas para controlar bucles, los cuales se ejecutarán mientras un determinado evento no haya ocurrido.
El valor del indicador se inicializa normalmente a falso antes de la entrada al bucle y se actualiza a verdadero cuando un evento específico ocurre dentro del cuerpo del bucle. Un bucle controlado por bandera o indicador se ejecuta **mientras no se produce ese evento específico.**

EJEMPLO: Leer y contar caracteres **mientras no tipeen** un carácter dígito.

```
package while03;
// muestra caracteres tipeados por el operador
// finaliza cuando se introduce un dígito
public class Main {
    public static void main(String[] args){
        boolean digito_leido = false; //  bandera
        char car;
        while(!digito_leido){
            System.out.println("Un caracter, por favor ");
            car = In.readChar();
            digito_leido = '0' <=car && car <= '9';
            System.out.println("leimos: "+car);
        }
        System.out.println("Demo terminado!!!");
    }
}
```

```
run:
Un caracter, por favor
A
leimos: A
Un caracter, por favor
B
leimos: B
Un caracter, por favor
C
leimos: C
Un caracter, por favor
1
leimos: 1
Demo terminado!!!
```

```
package while04;
// lee caracteres tipeados por el operador
// mientras no superen una cantidad parámetro
public class Main {
    public static void main(String[] args){
        int cant = 0;
        int cont = 0;
        System.out.println("Cuantos caracteres? ");
        cant = In.readInt();
        boolean leidos_sufi = false;     // bandera
        char car;
        while(!leidos_sufi){
            System.out.println("Un caracter, por favor ");
            car = In.readChar();
            cont++;
            leidos_sufi = cont >= cant;4
            System.out.println("leimos: "+car);
        }
        System.out.println("Demo terminado!!!");
    }
```

```
run:
Cuantos caracteres?
4
Un caracter, por favor
a
leimos: a
Un caracter, por favor
V
leimos: V
Un caracter, por favor
2
leimos: 2
Un caracter, por favor
U
leimos: U
Demo terminado!!!
```

```
}
```

Sentencias de quiebre de control.

Tenemos dos sentencias que alteran el flujo "normal" de las estructuras de control

- break. Posibilita la salida abrupta de un bucle o alternativa selectiva.
- continue. Retorna el flujo a la condición de control del bucle.

La forma mas rápida de entenderlas es mediante un ejemplo.

- Supongamos que queremos ir mostrando números en una secuencia ascendente, sólo que estos números son generados aleatoriamente. Por lo tanto deberemos descartar los que no pertenecen a la secuencia pedida. Usaremos "**continue**" para ir a la condición de control del bucle, y como esta es verdadera, vamos a la primera sentencia del cuerpo: generación de otro número.

- La cantidad de números a mostrar es también determinado en forma random (al azar). Usaremos "**break**" para salir del ciclo cuando este requisito esté cumplido.

- El bucle que usamos será del tipo "eterno" o de duración indefinida.

```
package while05;
public class Main {
    public static void main(String[] args){
        int cant = 0;   //  Cuantos queremos
        int cont = 0;   //  Cuantos tenemos
        int desc = 0;   //  Cuantos descartamos
        int nro;        //  nro corriente generado al azar
        int ant  = 0;   //  nro anterior  a este corriente.
        cant = (int)(10*Math.random());  //    quiero cant términos
        System.out.println("Términos en secuencia");
        System.out.println("Cantidad: "+cant);
        while(true){
            nro = (int)(100*Math.random());    //    generado al azar
            if(cont == 0) ant = nro; //   primera vez, valuamos cant
            if(nro < ant){
                desc++;    //  fuera de secuencia, descartamos
                continue;  //   a generar otro
            }else ant = nro;  //  guardamos;
            if(++cont > cant) break;
            System.out.println("generamos: " + nro);
        }
        System.out.println("Descartamos "+desc);
        System.out.println("Demo terminado!!!");
    }
}
```

```
run:
Términos en secuencia
Cantidad: 2
generamos: 23
generamos: 82
Descartamos 19
Demo terminado!!!
```

En general, si podemos evitar el uso de estas sentencias, hagámoslo.

Existe un llamado Teorema Fundamental de la Programación Estructurada que enuncia taxativamente: "***Toda estructura sólo debe contener un único punto de entrada y un único punto de salida***". Como esto tiene su costo, los lenguajes posteriores a este paradigma brindaron las herramientas para burlar este teorema. Cada **break** que pongamos dentro de la estructura será un punto de salida adicional, cada **continue** uno adicional de entrada. Sin embargo, muchas veces esto no vale la pena, no simplifica la lógica.

El mismo enunciado del algoritmo anterior, sin usar **break** ni **continue**:

```
package while06;
public class Main {
    public static void main(String[] args){
```

```
        int cant = 0;   //  Cuantos queremos
        int cont = 0;   //  Cuantos tenemos
        int desc = 0;   //  Cuantos descartamos
        int nro;        //  nro corriente generado al azar
        int ant  = 0;   //  nro anterior  a este corriente.
        boolean terminado = false;
        cant = (int)(10*Math.random());  //    quiero cant términos
        System.out.println("Términos en secuencia");
        System.out.println("Cantidad: "+cant);
        while(!terminado){
            nro = (int)(100*Math.random());    //    generado al azar
            if(cont == 0) ant = nro; //   primera vez, valuamos cant
            if(nro < ant){
                desc++;    //  fuera de secuencia, descartamos
            }else{
                ant = nro;  //  guardamos;
                if(++cont >= cant) terminado = true;
                System.out.println("generamos: " + nro);
            }
        }
        System.out.println("Descartamos "+desc);
        System.out.println("Demo terminado!!!");
    }
}
```

```
run:
Términos en secuencia
Cantidad: 4
generamos: 90
generamos: 94
generamos: 97
generamos: 99
Descartamos 22
Demo terminado!!!
```

Repetición: El Bucle For
El ciclo for tiene **dos formas diferentes.**

La primera se conoce también como ciclo **foreach** y se usa exclusivamente para recorrer los elementos de una colección. A la variable del ciclo se le asigna el valor de los sucesivos elementos de la colección en cada iteración del ciclo.

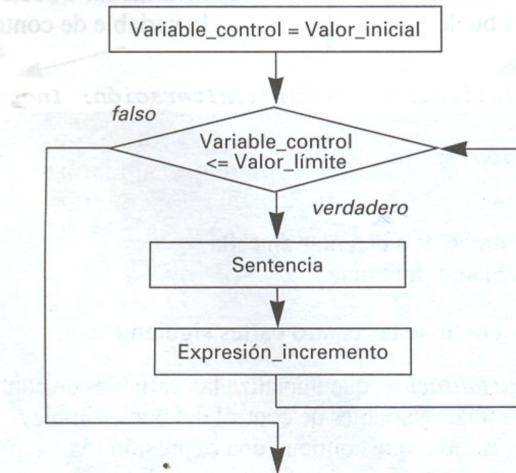
for (declaración-de-variable : colección) { sentencias }
Ejemplo: for (String nota : lista) { System.out.println(nota);}
Veremos en detalle su uso cuando estudiemos la implementación de arrays de tamaño variable mediante la clase de colección ArrayList

La segunda es heredada de C++. El bucle for de C++ es superior a los bucles for de otros lenguajes de programación tales como BASIC, Pascal y C, ya que ofrece más control sobre la inicialización e incrementación de las variables de control del bucle.
En esta el ciclo for ejecuta un bloque de sentencias tantas veces como la condición se evalúe verdadera. Antes de iniciar el ciclo, se ejecuta exactamente una vez, una sentencia de inicialización. La condición es evaluada antes de cada ejecución del cuerpo del ciclo (por lo que el cuerpo del ciclo podría no ejecutarse). Se ejecuta una sentencia de incremento al finalizar cada ejecución del cuerpo del ciclo.

La sentencia for (bucle for) es un método para ejecutar un bloque de sentencias un número de veces. El bucle for se diferencia del bucle while en que las operaciones de control del bucle se sitúan en un solo sitio: la cabecera de la sentencia.

Sintaxis

```
for (Inicialización;
      CondiciónPermanencia;
      Incremento)
            cuerpo
```



El bucle for que se estudia en esta sección es el más adecuado para implementar bucles controlados por contador, que son bucles en los que un conjunto de sentencias se ejecutan una vez por cada valor de un rango especificado, de acuerdo al algoritmo:

por cada valor de una variable_contador de un rango específico:
ejecutar sentencias

En esta segunda forma el bucle for contiene las cuatro partes siguientes:

- **inicialización**, (opcional) que inicializa las variables de control del bucle. Una o varias variables de control.
- **condición de permanencia en la iteración**, (opcional, si no informada es verdadera) que contiene una expresión lógica que hace que el bucle itere (repita) el cuerpo del bucle, mientras que la expresión sea verdadera.
- **incremento**, (opcional) que incrementa o decrementa la variable o variables de control del bucle.
- **Cuerpo** o bloque de acciones, (opcional) sentencias que se ejecutarán por cada iteración del bucle.

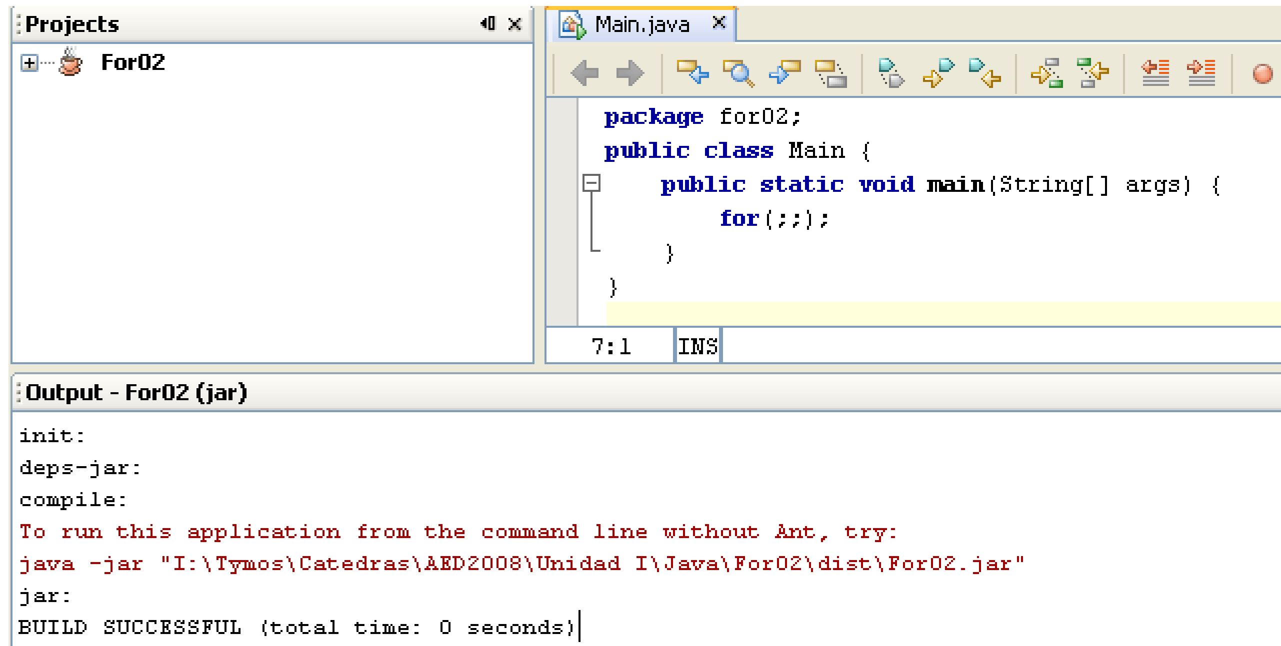
La sentencia for es equivalente al siguiente código utilizando while ➔

```
inicialización;
while (condiciónIteración){
      sentencias del bucle for;
      incremento;
}
```

```
package for01;
/**
 * Imprimir Hola un número variable (random) de veces
 * @author tymos
 */
public class Main {
    public static void main(String[] args) {
        for(int cant = (int)(10*Math.random()), i = 0;
            i<=cant; i++)
            System.out.println("Hola Nro "+i);
       System.out.println("demo terminado");
    }
}
```

```
run:
Hola Nro 0
Hola Nro 1
Hola Nro 2
Hola Nro 3
Hola Nro 4
demo terminado
```

Experimentemos un poco el bucle for… Habíamos dicho que cada parte era opcional. Que dirá el compilador de Java si no usamos **ninguna parte**?



Al compilador le parece bien. Si ejecutamos este proyecto nos encontramos con un proceso que es la ejecución de un ciclo eterno, que no hace nada útil. Como salir de él?. Si usamos SO Windows con <ctrl Alt del> podemos salir, matando proyecto y entorno; en general los entornos permiten terminar procesos .

Usemos el ciclo for en una forma mas clásica, usando cada una de sus partes. Calculemos el factorial de un número, por ejemplo, 5.

```
package for03;
//  Calculo del factorial de un número
public class Main {
    public static void main(String[] args){
        int fact = 1;
        // Queremos el factorial de 5
        for(int i = 1; i <= 5; i++)
            fact = fact * i;
        System.out.println("El factorial de 5 es "+fact);
    }
}
```

```
run:
El factorial de 5 es 120
BUILD SUCCESSFUL (total time: 2 seconds)
```

Podríamos hacer lo mismo con un ciclo for sin cuerpo?
Probemos, traslademos el contenido del cuerpo a la parte de incrementación.

```
package for04;
//  Calculo del factorial de un número
public class Main {
    public static void main(String[] args){
        int fact = 1;
        // Queremos el factorial de 5
        for(int i = 1; i <= 5; i++, fact = fact * i);
        System.out.println("El factorial de 5 es "+fact);
    }
}
```

run:
El factorial de 5 es 720
BUILD SUCCESSFUL (total time: 1 second)

Naturalmente, no estamos de acuerdo con el resultado. Que ha ocurrido? Al hacer este traslado hemos ejecutado fact = fact * i una vez mas de las necesarias

run:
El factorial de 5 es 120
BUILD SUCCESSFUL (total time: 0 seconds)

(esto nos pasa por usar el for apartándonos de la norma, apartarse del camino recto tiene sus riesgos …); si allí modificamos la condición de permanencia para i < 5 tendremos ➔

Hagamos otro ejemplo incrementando un contador y decrementando otro. En este ejemplo acumulemos ambos contadores, terminando cuando ellos "se cruzan". El resultado pedido es el valor de la acumulación de ambos contadores y la cantidad de pasadas realizadas.

```
package for05; //  Incrementando y decrementando contadores
public class Main {
    public static void main(String[] args) {
        int pas, incr, acuIncr, decr, acuDecr;
        for(pas=0, acuIncr = 0, acuDecr = 0, incr = 0, decr = 10;
            incr < decr;
            incr++, decr--){
                acuIncr+=incr;
                acuDecr+=decr;
                pas++;
        }

        System.out.println("Despues de " +pas+ " pasadas tenemos:");
        System.out.println("acuIncr: "+acuIncr);
        System.out.println("acuDecr: "+acuDecr);
    }
}
```

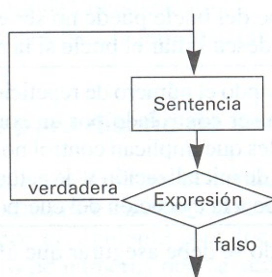
```
run:
Despues de 5 pasadas tenemos:
acuIncr: 10
acuDecr: 40
BUILD SUCCESSFUL (total time: 0 seconds)
```

Repetición: El Bucle do-while

La sentencia do-while se utiliza para especificar un bucle condicional que se ejecuta al menos una vez. Esta situación se suele dar en algunas circunstancias en las que se ha de tener la seguridad de que una determinada acción se ejecutará una o varias veces, pero al menos una vez.

Sintaxis

do sentencia while (expresión)



La construcción do comienza ejecutando sentencia. Se evalúa a continuación expresión. Si expresión es verdadera, entonces se repite la ejecución de sentencia. Este proceso continúa mientras expresión es verdadera.

Ejemplo introduciendo caracteres. El proceso continúa mientras el carácter sea dígito decimal.

```
package dowhile01;
public class Main {
    public static void main(String[] args) {
        char dig;
        do{
            System.out.println("Un dígito, por favor");
            dig = In.readChar();
        }while(dig>='0' && dig<='9');
        System.out.println("Demo finished ...");
```

```
run:
Un dígito, por favor
3
Un dígito, por favor
5
Un dígito, por favor
7
Un dígito, por favor
a
Demo finished ...
```

```
    }
}
```

Bucles Anidados

Es posible anidar bucles. Los bucles anidados constan de un bucle externo con uno o más bucles internos. Cada vez que se repite el bucle externo, los bucles internos se repiten, se reevalúan los componentes de control y se ejecutan todas las iteraciones requeridas.

EJEMPLO. Deseamos imprimir sucesivos productos x * y, para un rango de valores de cada uno de ellos; los limites de cada rango se definen inicialmente.

```
package cicloanid01;
public class Main {
   public static void main(String[] args) {
      int xIni = 4, xFin = 6, yIni = 5, yFin = 8, prod;
      System.out.println("Tabla de productos");
      System.out.println("entre valores de x "+ xIni+", "+xFin);
      System.out.println("y     valores de y "+ yIni+", "+yFin);
      for(int x = xIni; x <=xFin; x++){
         System.out.println("  valor de x = "+ x);
         for (int y = yIni; y <= yFin; y++){
            prod = x * y;  System.out.println("   * "+y+" = "+pro
         }
      }
      System.out.println("Demo terminado");
   }
}
```

```
run:
Tabla de productos
entre valores de x 4, 6
y     valores de y 5, 8
  valor de x = 4
   * 5 = 20
   * 6 = 24
   * 7 = 28
   * 8 = 32
  valor de x = 5
   * 5 = 25
   * 6 = 30
   * 7 = 35
   * 8 = 40
  valor de x = 6
   * 5 = 30
   sigue
```

Ejemplo: Escribir un programa que visualice un triángulo isósceles.

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

Una solución propuesta se realiza mediante un bucle externo y dos bucles internos. Cada vez que se repite el bucle externo se ejecutan los dos bucles internos. El bucle externo se repite cinco veces (cinco filas); el número de repeticiones realizadas por los bucles internos se basan en el valor de la variable fila. El primer bucle interno visualiza los espacios en blanco no significativos; el segundo bucle interno visualiza uno o más asteriscos.
La primera fila consta de cuatro blancos y un asterisco, la fila 2 consta de tres blancos y tres asteriscos, y así sucesivamente; la fila 5 tendrá 9 asteriscos
La cantidad de asteriscos responde a (2 x fila - 1).

```
package isosceles;
public class Main {
    public static void main(String[] args) {
        //  datos locales...
        int num_1ineas = 5;
        char blanco = ' ';
        final char asterisco = '*';

        System.out.println(" Isosceles");
```

```
run:
 Isosceles
    *
   ***
  *****
 *******
*********
BUILD SUCCESSFUL
```

```
        // dibujar cada línea: bucle externo
        for (int fila = 1; fila <= num_1ineas; fila++){

            //  imprimir espacios en blanco: primer bucle interno
            for (int blancos = num_1ineas -fila; blancos > 0; blancos--)
                System.out.print(blanco);

            //  imprimir asteriscos: segundo bucle interno
            for (int cuenta_as = 1; cuenta_as < 2 * fila; cuenta_as ++)
                System.out.print(asterisco);
            System.out.println();   //  Avanzo línea

        }   // fin del bucle externo
    }   //  main
}       //  Mai
```

Será posible imprimir este triangulo usando **un único bucle interno**, con un algoritmo mas simple?

Tenemos las formulas: (Relación entre datos y resultados)
cantidad de blancos: num_1ineas –fila
cantidad de asteriscos: 2 x fila - 1

Analizamos ➔ Estrategia de resolución

La cantidad total de caracteres es la suma de las expresiones anteriores.
Podemos idealizarlo así: en este único ciclo interno mandamos imprimir un carácter cada vez, inicialmente un espacio en blanco y luego '*'.
Entonces todo se reduce a saber en que punto del ciclo modificamos el carácter De espacio en blanco para '*'. Vamos al ejemplo.

```
package isosceles01;
public class Main {
    public static void main(String[] args) {
        //  datos locales...
        int num_1ineas = 5;
        char caracter = ' ';
        int cantCar;    //  cantidad de caracteres a imprimir
        int cantBla;    //  cantidad de blancos
        System.out.println(" Isosceles");   //  avanzamos una linea

        // dibujar cada línea: bucle externo
        for (int fila = 1; fila <= num_1ineas; fila++){
            cantCar = cantBla = num_1ineas - fila;
            cantCar+= 2 * fila - 1;
            caracter = ' ';
            for (int pos = 1; pos <= cantCar; pos++){
                if(pos == cantBla+1)caracter = '*';
                System.out.print(caracter);
            }
            System.out.println();   //  Avanzo línea
        }   // fin del bucle externo
    }   //  main
}       //  Main
```

```
run:
  Isosceles
      *
     ***
    *****
   *******
  *********
BUILD SUCCESSFUL
```

Cual de los programas es el mejor?
Un buen criterio: el más simple.

Si lo vemos por el lado de la lógica, puede ser opinable. El segundo programa tiene una lógica más elaborada que el primero, dejemos que el alumno decida si eso lo hace más complicado o más simple.

Un criterio objetivo, veámoslo por el lado de la cantidad de sentencias. Sin hacer "trampa", (agrupar sentencias) contamos:

El primero consta de 20 líneas.
El segundo consta de 22 líneas.

Esta solución no pareciera ser mas simple, verdad? Pero el alumno puede probar otras.

Ejercicios propuestos para el alumno

1 ¿Cuál es la salida del siguiente segmento de programa?

```
for (int cuenta = 1; cuenta < 5; cuenta++)
      System.out.println(" "+cuenta*2);
```

2 Codifique el mejor bucle para las siguientes tareas:

a Suma de series tales como 1/2+1/3+1/4+1/5+...+1/50
b Lectura de la lista de calificaciones de un examen de Historia
c Visualizar la suma de enteros en el intervalo 11...50.

3 Considerar el siguiente código de programa:

```
int i = 1;
while (i <= n)

      if ((i%n) == 0)
            ++i;
System.out.println("Valor de i " + i);
```

a ¿Cuál es la salida si n es O?
b ¿Cuál es la salida si n es 1?
c ¿Cuál es la salida si n es 3?

4 Escriba un programa que calcule y visualice

1 + 2 + 3 + ... + (n-l) + n

donde n es informado por teclado

5 ¿Cuál es la salida del siguiente bucle?

```
 suma = O;
 while (suma < 100) suma += 5;
System.out.println("Valor de suma " + suma);
```

6 Escribir un bucle while que visualice todas las potencias de un entero n, menores que un valor limite informado.

7 ¿Qué hace el siguiente bucle while? Reescribirlo con sentencias for y dowhile.

```
 num = 10;
 while (num <= 100)(
      System.out.println("num: "+num);
 num += 10;
}
```

8 Para m = 3, n = 5, ¿cuál es la salida de los siguientes segmentos de programa?

```
A    for (int i = 0; i < n; i++)(
            r (int j = 0; j < i; j++)
            stem.out.print("*")¡
```

```
                  stem.out.println();
            }
      B     for (int i = n; i > O; i--){
                  r (int j = m; j > 0; j--)
                  stem.out.print("*")¡
                  stem.out.println();
            }
```

9 Escribir un programa que visualice el siguiente dibujo:

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

10 Escribir un programa que lea una línea de texto y a continuación visualice el último carácter leído. Así, para la entrada “Cuarto menguante”, la salida es ‘e’