Ingeniería en Desarrollo de Software
3er semestre

Programa de la asignatura:
Programación orientada a objetos I

Clave:
Ingeniería: **TSU:**
15142316 / 16142316

Universidad Abierta y a Distancia de México

Índice

Unidad 1. Introducción a Java

Presentación de la unidad

En esta primera unidad de la asignatura Programación orientada a objetos I (POO I) aprenderás el concepto de la programación orientada a objetos y la descripción de los elementos que la integran, así como del lenguaje a implementar, que será Java, sus características y la estructura de un programa.

Propósitos

Al término de esta unidad lograrás:

- Distinguir entre programación orientada a objetos y programación estructurada.
- Reconocer las características de la programación orientada a objetos.
- Determinar las características y especificaciones de la programación en Java.
- Identificar la organización de un programa en Java.
- Desarrollar programas modulares.

Competencia específica

Desarrollar un programa básico para la solución de problemas matemáticos simples tomando en cuenta el entorno, características y especificaciones de los programas Java a través de dicho lenguaje.

1.1. Características de la programación orientada a objetos

En la actualidad se puede entender que en la vida diaria las personas se rodean de objetos y que estos objetos coexisten con ellas, pero ¿qué es un objeto?

Se puede describir y nombrar los objetos para entender cuáles son las diferencias entre ellos, por ejemplo si vas a un supermercado, encontrarás “carritos de super” y canastillas de mercado, ambos son objetos con propiedades o atributos en común, espacio para cargar o transportar artículos, sin embargo, al mismo tiempo tienen diferencias, la canastilla de mercado se necesita cargar, pero al “carrito de super” no, es decir cada uno es para un fin específico y práctico.

Objeto.
Imagen tomada de http://pixabay.com/es/caddy-carro-de-compras-161016/

De tal modo, se puede definir a un objeto como una entidad compleja con propiedades (datos, atributos) y comportamiento (funcionalidad, métodos). Con todas estas características es conveniente afirmar que todo lo que rodea a ser humano se puede representar y que cada uno de estos objetos posee una interfaz que especifica cómo éstos pueden interactuar con otros, dichos objetos tienen instrucciones propias (Joyanes, 2001).

1.1.1. Abstracción

De acuerdo con Joyanes, la abstracción es:

> La propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos que no son significativos. Durante el proceso de abstracción es cuando se decide qué características y comportamiento debe tener el modelo. Un medio de reducir la complejidad es la abstracción. Las características y los procesos se reducen a las propiedades esenciales, dejando de lado los demás detalles no sustanciales. De este modo, las características complejas se hacen más manejables (Joyanes, 2001, p. 5).

La abstracción como tal sirva para poder pasar del plano material al plano mental. Ahora bien, ¿cómo se relaciona la abstracción con la POO? Imagina que tienes un grupo de tornillos y de dicho grupo un tornillo se podría manejar con un desarmador específico, pero también con cualquier otro desarmador, aunque no sea propio. Lo mismo pasa en un sistema software, los módulos son independientes, pero a la vez funcionan como un gran modulo del software, es decir se puede partir un gran software para poder hacerlo en módulos, esto hará más fácil la identificación de sus características básicas.

Metáfora de los POO.
Imagen tomada de
http://pixabay.com/es/herramienta-destornillador-llave-78016/

1.1.2. Polimorfismo

"La propiedad de polimorfismo es aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase" (Joyanes, 2001, p. 6). Dicho de otro modo es la capacidad que tiene una clase para adaptarse a diferentes usos sin necesidad de modificarla. Así, por ejemplo, la operación copiar puede darse en diferentes clases: copiar un libro, copiar un archivo, copiar una actitud, copiar un comportamiento, copiar en un examen, etc. Siempre se ejecuta una operación distinta pero con el mismo nombre "copiar". Por tanto:

El polimorfismo es la propiedad de una operación de ser interpretada exclusivamente por el objeto al que corresponde.

1.1.3. Encapsulación

Si bien externamente los objetos pueden parecer iguales, internamente pueden ser muy distintos. Por ejemplo, si sabes manejar un reproductor de DVD, puedes manejar cualquier otro reproductor como CD y BLUE RAY, esto es encapsular, los dos reproductores pueden ser parecidos por fuera, pero por dentro funcionan de diferente forma.

Reproductor de DVD.
Imagen tomada de
http://pixabay.com/es/reproductor-de-dvd-dvd-de-audio-28885/

Esta propiedad permite intercambiar objetos en una operación; retomando el ejemplo de los reproductores, se puede cambiar la carcasa por una más moderna o llamativa sin que el funcionamiento interno se vea afectado.

1.1.4. Herencia

La herencia en el paradigma de la POO es un mecanismo que se utiliza para efectuar la reutilización del software, de modo tal que los programadores pueden crear clases nuevas a partir de clases ya existentes, tomándolas como base y creando una jerarquía de clases, de esta manera no se tiene que rehacer todo el código de una clase.

Si se ve de manera más sencilla, la herencia en programación funciona de manera similar a la herencia familiar, por ejemplo, si un papá tiene un hijo que se parece mucho físicamente con él, entonces ambos comparten ciertas características, se puede decir entonces que el papá le heredó sus características físicas al hijo.

Las características de un objeto pueden ser heredadas del objeto que está en el nivel superior, por ejemplo un novillo tiene orejas y cola y su pelo es negro como el de un toro normal, pero su cornamenta es más delgada y más fina, el toro y el novillo son parecidos pero no son iguales.

En programación la herencia se aplica sobre las clases, y hay clases padre, de las cuales de despliegan las clases hijo, que heredará las características del padre y pueden tener otras características más.

Hasta este punto se ha visto sólo los conceptos generales que dan cabida a la POO, más adelante en el curso se verá cómo estos conceptos se implementan ya en la programación.

1.1.5. Programación orientada a objetos vs estructurada

Desde la concepción de la programación estructurada (y por su misma etimología) se ha entendido de manera que se debe realizar de manera secuencial y haciendo el diseño como lo marcan los cánones: de arriba a abajo (Top-Down) o de abajo a arriba (Bottom-Up). De esta manera, se comienza a pensar y escribir una secuencia de instrucciones que deberán resolver el problema que se presenta y para el cual se escribe el programa deseado. Cuando la complejidad misma inherente al problema crece, por cualquiera que sea la razón, esta concepción deja de ser útil e inevitablemente se debe voltear la mirada a otro paradigma de programación, que, no importando cuán grande y complejo se torne la solución del problema, sea capaz de manejarlo e implementarlo, se está hablando de la programación orientada a objetos (POO).

La POO tiene sin duda muchas de sus bases en la programación estructurada, la principal diferencia entre estos dos paradigmas radica en que en la programación estructurada la atención principal del programa (y del programador) está en la secuencia de las instrucciones, mientras que en la POO la atención principal está en las partes (objetos) que la conforman, cómo interactúan y las operaciones que se pueden hacer sobre éstos. Para tener un panorama más claro sobre lo explicado se sugiere imaginar un archivo electrónico cualquiera. En términos de la POO las acciones que se aplican sobre este objeto pueden ser imprimir, borrar, renombrar, escribir más letras, entre otros. Aunque se piense que si se compara el código generado para las mismas acciones en programación estructurada y POO sea igual, no lo es; la principal diferencia estriba en que para la POO estas operaciones (métodos) conforman al objeto en sí, son parte de él y en la programación estructurada son funciones aisladas que no tienen vínculo alguno, más que desde donde son llamadas.

A continuación se presentan los siguientes puntos como principales diferencias entre ambos paradigmas:

- La POO se considera la evolución de la programación estructurada en cuanto a la atención de problemas complejos y cambiantes.
- La POO se basa en lenguajes que soportan descripción y escritura de tipos de dato complejos (no sólo primitivos) y las operaciones que se pueden efectuar sobre ellos, la clase.

- La POO incorpora niveles profundos de abstracción que permiten utilizar (entender y aplicar) conceptos como el polimorfismo, la herencia, la sobrecarga, entre otros.

Los creadores de la programación estructurada encontraron solución al problema presentado por ésta (no ser apta ante la complejidad de los problemas a resolver), fue el seguimiento de rigurosos métodos de trabajo al utilizar la programación estructurada, pero pocos fueron quienes aceptaron trabajar bajo tal presión.

Teclado.
Imagen tomada de
http://pixabay.com/es/photos/?q=teclado+pc
+&image_type=illustration&cat=&order=best

Sin embargo los problemas no son inherentes a la programación estructurada, sólo es mala la aplicación de sus conceptos a la resolución de problemas, se listan a continuación una serie de malas prácticas de la programación estructurada:

- No se sigue completamente la forma en que trabaja el cerebro del ser humano, ya que éste capta la realidad con base en “objetos” y sus propiedades (atributos) y las cosas que puede hacer o que se hacen sobre él (método).
- Dificultad para extender los programas existentes a nuevas necesidades.
- No se centra el modelo en el mantenimiento, se centra en la generación de soluciones repetitivas para los mismos problemas.
- Las entidades funcionales no se sincronizan o lo hacen de manera difícil y tortuosa.

Muchos lenguajes de programación que se dicen orientados a objetos aún aceptan gran parte del paradigma de la programación estructurada, que da pie a hacer una programación mezclada y llevando lo peor de ambos mundos. Se recomienda evitar esto.

Dirígete al aula virtual y atiende las indicaciones del (de la) Docente en línea para resolver las actividades 1 y 2.

1.2. Características del lenguaje Java

Programación Java.
Imagen tomada de
http://pixabay.com/es/java-copa-caf%C3%A9-programaci%C3%B3n-151343/

Hasta el momento en que se escribe el presente documento hay varias versiones que describen el origen de Java como proyecto y como lenguaje de programación, y que van desde situar sus orígenes entre finales de la década de los ochentas y 1990 (Zukowski, 1997) hasta mediados de la década de los noventas. De manera general se describe como un lenguaje de programación que fue pensado para desarrollar software que se utilizara dentro de la electrónica de consumo (refrigeradores, lavadoras, entre otros) desarrollado por el *Green Team de Sun Microsystems,* liderado por Bill Joy, y aunque la idea original de usarlo en electrónicos fracasó, fue rescatado tiempo después para ser usado como lenguaje de propósito general, y esta idea le dio la principal característica que lo hizo popular y lo sigue manteniendo en el gusto generalizado: su capacidad multiplataforma. Además, con las API que dan soporte a acceso a base de datos, objetos remotos y modelos de componentes de objetos; internacionalización, impresión y reporteo, encriptación, procesamiento de señales digitales y muchas otras tecnologías y capacidades lo posicionan tomando gran parte de la tajada sobre los lenguajes de programación más utilizados.

1.2.1. Generalidades de Java

En una de las primeras publicaciones que hacen referencia a Java (Zukowski, 1997), Sun lo describe como un lenguaje sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto rendimiento, multihilo, y dinámico .

Sus creadores reconocen que estas palabras pueden llegar a ser pretenciosas, pero el hecho es que, en su totalidad, se describe acertadamente el lenguaje. Para tener una mejor comprensión de la descripción hecha sobre Java, se revisarán las características del lenguaje detrás de cada palabra.

Sencillo. Java es un lenguaje simple. La mayoría de los lenguajes de programación orientados a objetos no son ni cercanamente sencillos ni fáciles de utilizar o comprender, pero Java es un poco más fácil que C++, que se consideraba como el lenguaje de programación más popular hasta la implementación de Java. Java ha puesto simplicidad en la programación en comparación con C++, incorporando características medulares de C++ y eliminando algunas de éstas que hacen de C++ un lenguaje difícil y complicado.

Haciendo referencia a la facilidad de Java y su sencillez, se dice que es simple porque consta sólo de tres tipos de datos primitivos: números, boolean y arrays. Todo en Java en una clase. Por ejemplo, las cadenas son objetos verdaderos y no arrays de caracteres. Otros conceptos que hacen la programación en C++ más complicada son los punteros y la herencia múltiple. Java elimina los punteros y reemplaza la herencia múltiple en C++ con una estructura única denominada interfaz (Joyanes y Fernández, 2001).

La utilización de asignación de memoria y recolección de basura automáticas son otro aspecto que abona a la sencillez de Java sobre otros lenguajes, por ejemplo con el ya citado C++, donde el programador debe realizar estas tareas de forma manual, con los posibles errores que esto puede conllevar. Otra facilidad que proporciona Java es la elegancia de su sintaxis que hacen más fácil la lectura (comprensión) y escritura de programas.

Orientado a objetos. La programación orientada a objetos permite de una manera muy sencilla modelar el mundo real o los problemas que en él se presenten y necesiten ser resueltos mediante programas, cualquier cosa del mundo real puede ser modelada como un objeto. Un libro es un objeto, un automóvil es un objeto e incluso un préstamo o una tarjeta de crédito es un objeto. Un programa en Java se denomina orientado a objetos debido a que la programación en Java se centra en la creación, manipulación y construcción de objetos (Joyanes y Fernández, 2001).

Java es un lenguaje de programación orientado a objetos. Lo que significa que el trabajo real sobre la construcción de los programas se centra en un gran porcentaje (por no decir la totalidad) en los datos de la aplicación y los métodos que manipulan estos datos, en lugar de pensar estrictamente en términos de los procedimientos. Si se está acostumbrado a utilizar la programación basada en procedimientos (por ejemplo, con el lenguaje C o

BASIC), es posible que sea necesario cambiar la forma de diseñar los programas o aplicaciones cuando se utiliza Java. Una vez que se experimente lo poderoso que resulta este paradigma (orientado a objetos), pronto se ajustará a él.

Cuando se trabaja con el paradigma orientado a objetos, una clase es una colección de datos y tiene además métodos que operan sobre esos datos. En conjunto, los datos y métodos describen el estado y el comportamiento de un objeto.

Java cuenta con un amplio conjunto de clases organizadas en paquetes, que se pueden utilizar al programar. Proporciona paquetes de clases para la interfaz gráfica de usuario (java.awt), clases que se encargan de la entrada y salida (java.io), funcionalidad de soporte de red (java.net), entre muchos otros paquetes.

Desde la concepción inicial del diseño de Java se pensó en hacer de este un lenguaje totalmente orientado a objetos desde la base, a diferencia de otros lenguajes que fueron adoptando los lineamientos de este paradigma en sus características, como por ejemplo C++. La mayoría de los elementos disponibles en Java con objetos, con excepción de los tipos primitivos y los tipos booleanos. Las cadenas o arrays son en realidad objetos en java. Una clase es la unidad básica de compilación y ejecución en Java, todos los programas de Java cuentan con clases.

Interpretado. Java es un lenguaje interpretado y necesita un intérprete para ejecutar programas. El compilador de Java genera bytecode para la Máquina Virtual de Java (JVM), en lugar de compilar al código nativo de la máquina donde se ejecutará. El bytecode es independiente de la plataforma y se puede hacer uso de él en cualquier máquina (no necesariamente una computadora) que tenga un intérprete de Java.

En un entorno donde el lenguaje de programación es interpretado, la fase estándar de “enlace” casi se vería desvanecida. Si java cuenta con esta fase como un todo, es sólo porque el proceso de cargar clases nuevas en el ambiente de ejecución se hace precisamente en esta fase, que es un proceso incremental y mucho más ligero que se produce en tiempo de ejecución. Contrastando con el ciclo más lento y complicado de compilar-enlazar-ejecutar de lenguajes como C o C++. Los programas Java no necesitan

ser recompilados en una máquina destino. Se compilan en un lenguaje ensamblador para una máquina imaginaria, denominada máquina virtual (Joyanes y Fernández, 2001).

Arquitectura neutral y portable. Debido a que el lenguaje Java se compila en un formato de arquitectura propia o mejor dicho neutra llamada bytecode, un programa en Java puede ejecutarse en cualquier sistema, siempre y cuando éste tenga una implementación de la JVM y esta es la característica tal vez más notable que tenga Java.

Se pueden, por ejemplo, ejecutar applets desde cualquier navegador de cualquier sistema operativo que cuente con una JVM y más allá todavía, hacer sistemas autónomos que se ejecuten directamente sobre el sistema operativo. Esto es particularmente importante cuando se trabajará con aplicaciones distribuidas por internet o cualquier otro medio de distribución donde los usuarios no pueden (deben) tener un cierto sistema operativo funcionando en la computadora donde se ejecutará dicha aplicación.

En las aplicaciones que se desarrollan hoy en día, muy probablemente se necesite tener la misma versión ejecutándose en un ambiente de trabajo con UNIX, Windows o Mac. Más aún, con las diferentes versiones de procesadores y dispositivos (celulares, celulares inteligentes, consolas de video juegos, entre muchos otros) soportados por estos sistemas, las posibilidades se pueden volver interminables y la dificultad para mantener una versión de la aplicación para cada uno de ellos crece de igual manera, interminable.

La misma definición de bytecode hace que Java sea multiplataforma y no tenga la necesidad de complicados temas al portar la aplicación entre dispositivos como tipos de datos y su longitud, características y capacidades aritméticas; caso contrario, por ejemplo, del lenguaje C donde un tipo int puede ser de 16, 32 o 64 bits dependiendo de la plataforma de compilación y ejecución. Los programadores necesitan hacer un solo esfuerzo por terminar sus asignaciones, esto se puede lograr apegándose al slogan que hizo tan famoso Sun: “*Write Once, Run Anywhere*” (Escribe una vez, ejecuta donde sea).

Dinámico y distribuido. Java es un lenguaje dinámico. Cualquier clase construida en Java puede cargarse en cualquier momento dentro del ejecutor en cualquier momento. Así, si es cargada dinámicamente puede instanciarse de igual manera. Las clases en Java son

representadas por la palabra reservada class se puede obtener información en cualquier momento sobre ellas mediante el run-time.

En cualquier instante de la ejecución Java puede ampliar sus capacidades mediante el enlace de clases que esté ubicadas en la máquina residente, en servidores remotos o cualquier sitio de la red (intranet/internet). Caso contrario de lenguajes como C++ que hacen esto al momento de la compilación, después ya no se puede. Se puede extender libremente métodos y atributos de una clase sin afectar la ejecución corriente, las capacidades descritas se encuentran dentro del *API Reflections*.

Robusto. Java fue diseñado para ser un lenguaje de programación que genere aplicaciones robustas. Java no elimina la necesidad del aseguramiento de calidad en el software; de hecho es muy posible y probable tener errores al programar en Java. No elimina tampoco la mayoría de los errores que se comenten al utilizar cualquier lenguaje de programación. Sin embargo al ser fuertemente tipado se asegura que cada variable o método que se utilice corresponda en realidad con lo que el programador quiso utilizar y no que se escapen errores en conversión de tipos de dato, por ejemplo. Java requiere declaración explícita de métodos, cosa que no se permite en otros lenguajes, como en C. Java ha sido pensado en la fiabilidad, eliminando la mayoría (o todas) las posibles partes de otros lenguajes de programación propensas a errores, por ejemplo, elimina los punteros y soporta el manejo de excepciones en tiempo de ejecución para proporcionar robustez a la programación. Java utiliza recolección de basura en tiempo de ejecución en vez de liberación explícita de memoria. En lenguajes como C++ es necesario borrar o liberar memoria una vez que el programa ha terminado (Joyanes y Fernández, 2001).

Seguro. Uno de los aspectos más sobresalientes de Java es el de ser un lenguaje seguro, es especialmente interesante debido a la naturaleza de la distribución de aplicaciones en Java. Sin un aseguramiento el usuario no estará tranquilo de bajar y ejecutar un código en su computadora desde internet o cualquier otra fuente. Java fue diseñado con la seguridad como punto principal y tiene disponibles muchas capas que gestionan la seguridad de la aplicación que se escribe, bloqueando al programador si intenta distribuir código malicioso escrito en lenguaje Java.

Los programadores de Java no tienen permitido cierto tipo de acciones que se consideran altamente vulnerables o que el típico caso de ataque contra un usuario común, por ejemplo el acceso a memoria, desbordamiento de arreglos entre muchos otros.

Otra capa de seguridad dentro de Java es el modelo sandbox que hace una ejecución controlada de cualquier bytecode que llega a la JVM, por ejemplo si se lograr evadir la regla del código no malicioso al momento de compilar, este modelo en su ejecución controlada evitaría que las repercusiones lleguen al mundo real.

Por otro lado, otra posible solución al aspecto de seguridad en Java es que se añade una firma digital al código de Java, el origen del software puede establecerse con esta firma y utilizando criptografía se oculta esta firma para que sea inaccesible e inmodificable por cualquier persona. Si hay confianza en una persona específica de alguna organización, entonces el código puede ser firmado digitalmente por dicha persona, dando la seguridad que el resultado que se recibe sea de quien debe ser y no haya introducción de código por terceras personas ajenas a él.

Por supuesto la seguridad no puede entenderse ni manejar como una cosa que es totalmente blanca o totalmente negra, deben descubrirse los matices que pueda presentar y verificar todas las posibles vertientes que puedan tomar, así se asegura que todo está controlado. Ningún programa puede dar al 100% la garantía de la ausencia de errores, tampoco un ambiente de compilación o interpretación puede dar esta garantía. Java no se centra en la corrección de seguridad, se basa en la anticipación de los posibles errores que se puedan presentar.

Alto Rendimiento. Java es un lenguaje interpretado, por eso no será igual de veloz en la ejecución como un lenguaje compilado como C. Versiones tempranas de Java está aún decenas de veces más abajo que la velocidad de ejecución que proporciona C. Sin embargo, con el pasar del tiempo este aspecto ha ido mejorando sobre la base del compilador JIT (*Just in Time*) que permite programas en Java de plataforma independiente se ejecuten casi tan rápido como los lenguajes convencionales compilados.

Multihilo. Es fácil imaginar cómo funciona una aplicación que hace múltiples cosas a la vez, por ejemplo en un navegador web, donde, se hace la descarga al mismo tiempo del texto, imágenes, videos y demás componentes de las páginas que se visiten, pero esta descarga se hace por separado, donde una función específica se hace cargo de descargar las imágenes, otra el texto y así con cada uno de los compontes. Obviamente lo hacen al mismo tiempo, por esto se dice que es multihilo. Java es un lenguaje multihilo, ya que soporta la ejecución de múltiples tareas al mismo tiempo y cada uno de esos hilos puede soportar la ejecución de una tarea específica diferente. Un beneficio importante de esta característica multihilo es el aporte que da a las aplicaciones basadas precisamente en esto, ya que como se mencionó, incrementa el rendimiento de la aplicación, sobre todo en aplicaciones basadas en interfaces gráficas de usuario.

Aunado a lo anterior y haciendo referencia a la facilidad de Java, programar multihilo en Java es muy fácil, ya que si se ha tratado de trabajar con hilos en C o C++ se notará la enorme dificultad que esto representa. La clase Thread da el soporte, la facilidad y los métodos para iniciar y terminar la ejecución y hacer uso de los hilos, que a su vez se encuentra contenida en el nombre de espacio java.lang. La sintaxis del lenguaje Java de igual manera tiene soporte directo sobre la palabra reservada *synchronized,* la cual hace extremadamente fácil para marcar secciones de código o métodos completos que necesitan ser ejecutados de uno en uno o dicho de mejor manera “sincronizados”.

1.2.2. Máquina virtual de Java

¿Qué es la JVM?

Una Máquina Virtual de Java (JVM) es un software de proceso nativo, es decir, está desarrollada para una plataforma específica que es capaz de interpretar y ejecutar un programa o aplicación escrito en un código binario (el ya mencionado bytecode), que se genera a partir del compilador de Java.

En otras palabras, la JVM es la piedra angular del lenguaje de programación Java. Es el componente que hace que Java tenga su particular y muy mencionada característica

multiplataforma, generar un código compilado tan compacto y la capacidad del lenguaje de proteger a sus usuarios de código malicioso.

La JVM es una máquina de computación de tipo abstracto. Emulando a una máquina de componentes reales, tiene un conjunto de instrucciones y utiliza diversas áreas de memoria asignada para ella. La implementación del primer prototipo de la JVM, hecha por Sun Microsystems, emulando todo su conjunto de instrucciones fue elaborada sobre un hardware muy específico, lo que hoy se podría conocer como los modernos *Personal Digital Assistant* (PDA). Las versiones actuales de la JVM están disponibles para muchas (casi todas) las plataformas de computación más populares. Debe entenderse sin embargo que, la JVM no asume una tecnología específica o particular, esto es lo que la vuelve multiplataforma. Además, no es en sí interpretada ni sólo pueda funcionar de esta manera, de hecho es capaz de generar código nativo de la plataforma específica e incluso aceptar como parámetro de entrada, en lugar del bytecode, código objeto compilado, por ejemplo funciones nativas de C. De igual manera, puede implementarse en sistemas embebidos o directamente en el procesador.

Cada logotipo y marca pertenecen a sus respectivos dueños. La UnADM sólo los expone con fines educativos

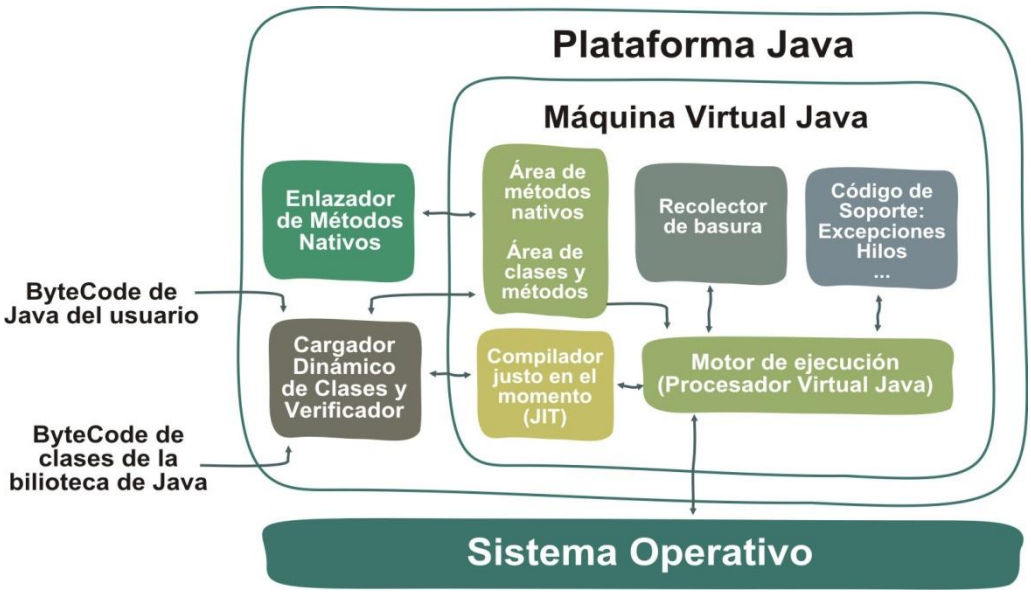
¿Cómo funciona la JVM?

La JVM no sabe nada del lenguaje de programación Java, no es de su interés, porque un compilador ya se encargó antes de ese proceso, sólo le interesa un formato específico de archivo, el formato de archivo de clase o entendido de otra manera, el bytecode, más una tabla de símbolos e información complementaria.

Haciendo una apología sobre la base de la seguridad, la JVM impone formatos fuertes y limitaciones estructurales en el bytecode (por ejemplo los apuntadores a memoria, que ya se han mencionado en apartados anteriores). Sin embargo, cualquier lenguaje formal, con la funcionalidad que se pueda expresar en términos de un archivo de bytecode válido, puede ser organizado, interpretado y ejecutado por la JVM. Atraídos por la gran característica multiplataforma que imprime la JVM al lenguaje Java, los programadores de

otros lenguajes de programación han recurrido a ella como vehículo de entrega (ejecución) de estos lenguajes, que da a entender que su capacidad multiplataforma puede extender la implementación de casi cualquier lenguaje que cumpla con sus reglas para ser multiplataforma.

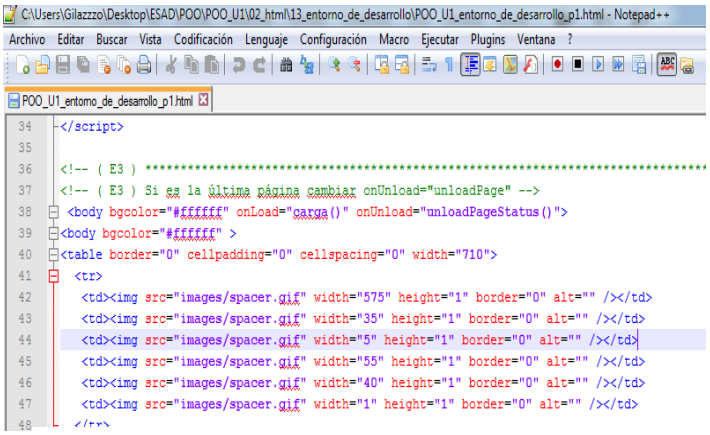
La funcionalidad completa, en forma de esquema se presenta en la siguiente figura:



Funcionalidad de la Máquina Virtual de Java. Tomada de Zukowski (1997).

1.2.3. Entorno de desarrollo y configuración

Cuando se está inmerso dentro del mundo de desarrollo de software en Java, hay varias opciones sobre la manera en que se pueden generar aplicaciones. Por ejemplo, se puede utilizar cualquier editor de texto (*notepad*) para poder capturar el código fuente y guardarlo en el disco duro con el formato y la extensión correctos, para después, en una segunda parte, compilar dicho código fuente y que arroje el bytecode a través de la línea de comandos haciendo una invocación al compilador de Java. Obviamente se deberá tener configurado en la estación de trabajo la manera en que ésta reconozca qué se está haciendo y de dónde se está llamando.



Desarrollo de plantilla

Para cualquier (o casi cualquier) lenguaje de programación, afortunadamente incluido Java, hay una serie de herramientas que facilitan enormemente poder producir aplicaciones completas y sólo requieren el esfuerzo del programador en las partes importantes del sistema, dejando a estas herramientas la "preocupación" de las cosas menos importantes. Para ejemplificar lo anterior, se deberá entender de la siguiente manera: quien desarrolla sólo deberá preocuparse por los datos que involucren a su aplicación, la manera de cómo obtenerlos, procesarlo y tal vez almacenarlos, preocuparse por entender las reglas del negocio al cual pretende generar dicha aplicación, cómo y en qué parte afectan estas reglas para su aplicación y sus datos. La mejor manera de explotar el conocimiento que genere su aplicación y no estar preocupado por configurar variables de entorno, preocupado por configurar conexiones a bases de datos ni aspectos específicos o variables complejas de los servidores de aplicaciones web, entre muchas otras cosas inherentes al sistema, no a la aplicación.

Las herramientas antes mencionadas, cuando se trabajan en conjunto como una sola unidad operativa, se le conoce como Entorno Integrado de Desarrollo (EID) o IDE por sus siglas en inglés (*Integrated Development Environment*), que se pueda entender como un programa de tipo informático que conjunta al mismo tiempo varias herramientas útiles para la programación de aplicaciones. El IDE puede estar enfocado en un lenguaje específico o soportar una colección de ellos, como es usual en los IDE's modernos; por ejemplo el IDE más popular de Microsoft es el Visual Studio en su versión 2010 (al momento de escribir este documento) y tiene un soporte nativo para una variedad muy amplia de lenguajes soportados, e inclusive que se pueden mezclar. Cuando se transporta el concepto del IDE a Java, existen de igual manera varios de donde se puede hacer una elección, por ejemplo:

- Eclipse
- Borland JBuilder
- Sun Java Studio Creator

- NetBeans
- IBM WebSphere Studio Site Developer for Java
- Dreamweaver
- WebLogic Workshop
- Oracle JDeveloper
- IntelliJ Idea
- JCreator

Nota: Todas las marcas o nombres de productos son propiedad de sus respectivos dueños y sólo se usan para ilustrar el ejemplo expuesto.

Componentes del IDE

Los elementos básicos de un IDE, pero no limitado a ellos, son:

- Un **editor de texto**, que es el programa o parte del IDE que permitirá crear, abrir o modificar el archivo contenedor del código fuente del lenguaje con el que se está trabajando. Puede proporcionar muchas ventajas sobre un editor de texto separado del IDE, ya que por lo general cuentan con un resaltado de sintaxis sobre las partes que conforman el código fuente de la aplicación que se esté construyendo.
- Un **compilador**, que es el programa o parte del IDE que permitirá traducir el código fuente escrito dentro del editor a lenguaje máquina (bytecode en el caso de Java). De tal manera que el programador pueda diseñar y programar una aplicación utilizando palabras, frases o conceptos muy cercanos al lenguaje que utiliza normalmente, y el compilador se encargará de traducir estas sentencias en instrucciones comprensibles por el procesador de la computadora.
- Un **intérprete**, que es el programa o parte del IDE que permitirá hacer la "interpretación" del código compilado (como en el caso de Java), en donde se toma como entrada el bytecode y el resultado es la ejecución de las instrucciones de lenguaje máquina o código objeto contenidas en esta estructura de clases. El compilador e intérprete pueden trabajar en conjunto como se explica en este apartado o ser procesos independientes para la ejecución de un programa.
- Un **depurador**, que es el programa o parte del IDE que permitirá probar y localizar errores dentro del código de la aplicación (mayormente de tipo lógico o de omisión)

haciendo una traza de la ejecución completa del código y señalando dónde es un posible error del código o vulnerabilidad no considerada.

- Un **constructor de interfaces gráficas**, que es la parte del IDE que facilita una serie de componentes nativos del lenguaje con el que se trabaja (swing en Java) para poder ofrecer al usuario un ambiente amigable de tipo visual, en pocas palabras permite crear interfaces gráficas de usuario y complementarlas con el código fuente de nuestra aplicación.
- Opcionalmente la posibilidad de trabajar conectado a un **servidor** de control de versiones, que permitirá llevar un control preciso de cada versión de cada archivo que comprenda la aplicación, es particularmente útil cuando se trabaja con aplicaciones de tipo empresarial que conllevan, inclusive, miles de archivos de código fuente, y que al estar trabajando en grupo se vuelve indispensable esta funcionalidad que ofrece el IDE.

Configuración del IDE

Cuando se hace la instalación de un IDE la configuración que trae por defecto es la habitual y será suficiente para poder comenzar a trabajar sobre él, la personalización que se haga de las distintas herramientas con las que cuenta dependerá del gusto de cada programador. Las posibles configuraciones que se hacen sobre el IDE van desde cosas tan simples como el tipo de letra y su tamaño para mostrar el código fuente, hasta opciones muy avanzadas donde se consumen servicios web de determinado sitio de internet.

Otra posible opción de configuración interesante sobre cada proyecto que se maneja en el IDE es de qué parte toma las clases base (independientes a las que proporciona Java) y cómo será la salida de la aplicación (jar, jar compimido, entre otros).

1.2.4. Tipos de datos soportados en Java

Tipo de dato	Tamaño en bits	Rango de valores	Descripción
Números enteros			
Byte	8-bit complemento a 2	-128 a 127	Entero de un byte
Short	16-bit complemento a 2	-32,768 a 32,767	Entero corto
Int	32-bit complemento a 2	-2,147,483,648 a 2,147,483,647	Entero
Long	64-bit complemento a 2	-9,223,372,036,854,775,808L a 9,223,372,036,854,775,807L	Entero largo
Números reales			
Float	32-bit IEEE 754	+/- 3.4E+38F	Coma flotante con precisión simple
Doublé	64-bit IEEE 754	+/- 1.8E+308	Coma flotante con precisión doble
Otros tipos			
Char	16-bit Carácter	Conjunto de caracteres Unicode ISO	Un solo carácter
Booleano	true o false	Verdadero o falo	Un valor booleano
String	Variable	Conjunto de caracteres	Cadena de caracteres.

La manera de identificar qué tipo de dato utilizar es analizar qué datos almacenará esa variable para definir su tipo de datos con base en el rango de valores y la descripción dados en la tabla anterior.

Para utilizar un dato se debe colocar al inicio el tipo de dato que se quiera, seguido del nombre que tendrá ese dato y un punto y coma si no se quiere dar un valor inicial, pero si se va a dar un valor se coloca una asignación con el signo igual y el valor.

Ejemplo de utilización

```
public class DataTypes
{
      public static void main(String[] args)
      {
```

```
      int x;                // un dato declarado pero sin valor
inicial.

      boolean isReal=true;  // Los nombres son sensibles a
                            // mayúsculas y minúsculas,
                            // deben empezar por una letra y
                            // pueden contener números,_,$
      byte a = 121;         // Deben ser inferiores a 126
      short b = -10000;     // Deben ser inferiores a 25000
      int c = 100000; // Deben ser inferiores a 2100 mill.
      long d = 999999999999L; // Deben poner L al final
      float e = 234.99F;    // Deben ser < 3E38; F al final

      double f = 55E100;
      char charValue= '4';  // char '4' no es el entero 4

      //Las cadenas (strings) son objetos, no primitivos.
      //Ejemplo:
      String institucion = "esad";

      }
 }
```

Dirígete al aula virtual y atiende las indicaciones del (de la) Docente en línea para resolver la actividad 3

1.2.5. Operadores aritméticos, lógicos y relacionales

En el entorno Java, la palabra operador se le llama a un símbolo específico que realiza una "operación" determinada entre dos o más valores, se puede entender con el procedimiento habitual que se utiliza para sumar, por ejemplo:
3+5=8

El operador de suma (representado por el símbolo +) está actuando sobre el 3 y el 5 para producir el 8. Como se puede ver, la evaluación de los operadores y sus operandos siempre devuelve un valor, pues es la parte que define a las operaciones.

Pueden tener notación prefija o posfija, que se defina en la posición que se coloquen los operadores respecto a sus operandos, así en la notación prefija los operadores irán a la izquierda del operando (++a) y en la notación posfija los operadores irán a la derecha de los operandos (a++).

Existen varios tipos de operadores que se puede utilizar de distinta manera y que arrojan resultados diferentes para la realización de operaciones, a continuación se listan.

Operadores aritméticos

Operadores	Significado	Asociatividad
++	Incremento	Derecha a izquierda
--	Decremento	Derecha a izquierda
+	Unario + (símbolo positivo)	Derecha a izquierda
-	Unario – (símbolo negativo)	Derecha a izquierda
*	Multiplicación	Izquierda a derecha
/	División	Izquierda a derecha
%	Resto (módulo)	Izquierda a derecha
+	Suma	Izquierda a derecha
-	Resta	Izquierda a derecha

Utilización de operadores aritméticos

```
public class opAritmeticos
{
      public static void main(String[] args)
       {
            int j, k, m;
            int d= 123;
            j= d--; // j vale 122 y d vale 123
            System.out.println("j= " + j);
            k= ++d; // k vale 124 y d vale 124
            System.out.println("k= " + k);
            m= --d; // m vale 123 y d vale 123
            System.out.println("m= " + m);
            m= k % j; // operador Resto para los tipos int

            // k=124 y j=122, por tanto,
            m= 2 System.out.println("m= " + m);
            j= 5; k= 3; m= j/k; // División entera: m= 1
            System.out.println("m= " + m);
            System.exit(0);
```

```
        }
}
```

Operadores lógicos

Operador	Descripción
==	Igual (comparación de igualdad)
>	Mayor que
<	Menor que
&&	Conjunción lógica (and)
!=	Distinto
>=	Mayor o igual que
<=	Menor o igual que
||	Disyunción lógica (or)

Utilización de operadores lógicos

```
double c= 0.0, b= 3.0;
if (c != 0.0 && b/c > 5.0){
      System.out.println("se cumple la condición solicitada");
}
```

Operadores de asignación

Se deberá, en primer lugar, hacer la distinción entre la acción de asignación y el operador de comparación (igualdad), aunque aparentemente al inicio pueda resultar confuso por el parecido entre ambos, lo que distingue uno de otro es precisamente el concepto que cada uno de ellos representa y la forma de presentarse, así pues:

El operador "=" no es lo mismo que el operador "=="

El operador básico al utilizar la asignación es "=", que se utiliza para asignar un valor a otro. Como sigue:

int contador = 0;

Iniciando la variable *contador* con un valor de *0.*

Además de las facilidades ya descritas que proporciona el lenguaje Java, proporciona una forma abreviada de representar este tipo de operadores, con la finalidad de agilizar la escritura para los programadores. Se describen en la siguiente tabla:

Operador	Uso	Equivalente a

+=	a += b	a = a + b
-=	a -= b	a = a – b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
&=	a &= b	a = a & b

Operadores de bit

Dan la posibilidad de manipular los bits de los datos, especialmente el desplazamiento.

Operador	Uso	Operación
>>	a >> b	Desplaza los bits de “a” a la derecha “b” veces
<<	a << b	Desplaza los bits de “a” a la izquierda “b” veces
>>>	a >>> b	Desplaza los bits de “a” a la derecha “b” veces (omitiendo el signo)

La función específica de este tipo de operadores es desplazar los bits del operando ubicado a la izquierda de la expresión el número de veces que indique el operador de la derecha. La dirección del operador indica hacia donde correrán los bits. La siguiente sentencia da a entender bien el concepto que se trata de implementar, al desplazar los bits del entero 25 a la derecha de su posición:

25 >> 2

El número entero 25 está representado por el número 11001 en notación binaria, al aplicarle el operador de desplazamiento el número 11001 ahora se convertirá en 110 o el número entero 6. Es importante notar que los bits que se desplazan a la derecha se pierden al efectuar esta operación.

Precedencia de operadores

Java asigna (como se hace en las operaciones matemáticas normales) una importancia muy alta a la precedencia de los operadores. En la siguiente tabla se lista esta precedencia y entre más alta sea su posición, más importancia tendrá:

Operadores	Representación
Operadores posfijos	[], (), a++, a--

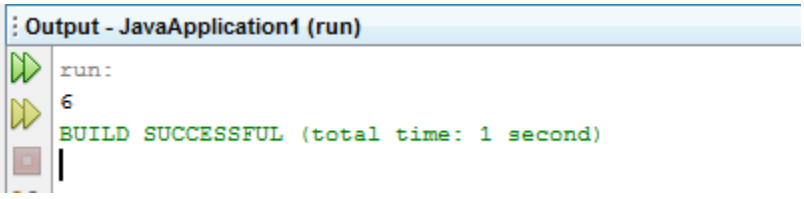
Operadores unario	++a, --a, +a, -a, ~, !
Creación o conversión	new(tipo) a
Multiplicación	*, /, %
Suma	+, -
Desplazamiento	<<
Comparación	==
Igualdad	==, !=
AND a nivel de bit	&
OR a nivel de bit	^
XOR a nivel de bit	|
AND lógico	&&
OR lógico	||
Condicional	? :
Asignación	=, +=, -=, *=, /=, &=, ^=, |=, <<===

Tabla de precedencia de los operadores

1.2.6. Conversión de tipos de datos

La conversión de tipos de datos (conocida también como *casting*) es pasar un objeto de un tipo en otro. Se puede ejemplificar de manera clara pensando en que si tienes un pequeño programa que tiene dos números enteros, el resultado de la división se almacena también en un entero, y quieres dividirlos, el resultado se truncará a un entero también, pues sólo estás tomando en cuenta la capacidad de almacenamiento de los enteros. El código y resultado sería como se muestra a continuación:

```
public class EjamploCast {
 public static void main(String[] args) {
 int di=33;
 int dv=5;
 int c=di/dv;
 System.out.print(c+ "\n");
 }
}
```

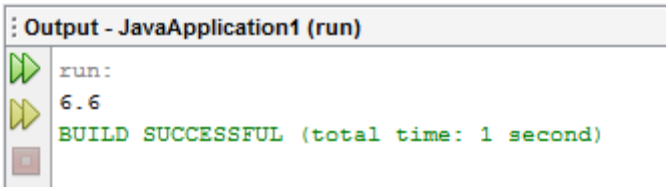


Como se ve, el resultado se truncó hasta el valor entero.

Pero si en el resultado es importante tomar en cuenta también los decimales para tener un resultado más preciso, se debe convertir el entero a flotante, eso se realiza de la siguiente manera.

```
float c= (float) di/dv;
```

Colocando el tipo de dato al que se convertirá entre paréntesis, ahora nuestro resultado mostrará los decimales que se necesite.



En este caso el resultado si tiene los decimales que se necesitan.

En un ejemplo más amplio si se tienen 3 clases A, B y C que se pueden almacenar en un arreglo de la siguiente manera:

```
Object [] array = new Object[3];
array[0] = new A();
array[1] = new B();
array[2] = new C();
```

Si se quieren usar estos objetos, se tendrá que hacer un cambio entre resultados que los tienes como array[i], que son Object, y no puedes llamar a métodos específicos de las clases A, B y C. La forma de recuperar las clases originales es con un cast, que al igual que el ejemplo anterior debes colocar el tipo al que quieres convertir entre parénesis antes del dato que se convertirá, entonces se pone el tipo A delante, entre paréntesis:

```
((A)array[0]).metodoDeA();
```

Obviamente no puedes convertir cualquier tipo a cualquier tipo. Si intentas convertir array[0] a B, te dará error, salvo que A herede de B. Pero estos casos más complejos se tratarán más adelante.

1.3. Organización de un programa

Java guarda una enorme semejanza con C/C++ por lo que no es difícil poder entender ni interpretar su código cuando se está familiarizado con estos lenguajes. De igual manera como Java sigue el paradigma de la programación orientada a objetos (POO) en código del lenguaje se construye a partir de la escritura de clases.

Durante esta unidad se sugiere que realices todos los ejemplos en el entorno de desarrollo para que lo tengas instalado y funcional desde esta primera unidad, ya que se estará utilizando a lo largo de toda la materia, además se sugiere que escribas, compiles y ejecutes los ejemplos de programas que se desarrollaron en esta unidad, para que se comprenda mejor lo visto.

1.3.1. Estructura general

Todo parte de una clase. Partiendo desde la concepción más simple de un programa en Java, este se puede escribir a partir de una sola clase y conforme la complejidad de la aplicación vaya en aumento, muy probablemente (de hecho lo es) el número de clases va a ir en aumento, se podría entender como una correlación directa. Por definición cada clase necesita un punto de entrada por donde comenzará la ejecución del programa, y la clase escrita debe contenerlo como el programa, rutina o método principal: main() y dentro de este método contiene las llamadas a todas las subrutinas o demás partes de la clase o incluso de la aplicación en general.

Una estructura simple, pero funcional de un programa en Java se expone a continuación:

```
public class inicio{
      public static void main(String[] args){
            Primer_sentencia;
            Sentecia_siguiente;
            Otra_sentencia;
            // …
            Sentencia_final;
} //cierre del método main
} // cierre de la clase
```

Al hacer la implementación del ejemplo anterior a algo un poco más real y palpable se sugiere como primer acercamiento un programa sencillo escrito en lenguaje Java que muestre en pantalla un mensaje de bienvenida a la programación en Java:

```
/**
* La clase bienvenida construye un programa que
* muestra un mensaje en pantalla
*/
public class bienvenida {
public static void main(String[] args) {
System.out.println("Hola, ");
System.out.println("bienvenido (a)");
System.out.println("a la programación en Java");
}
}
```

Para comprender mejor lo que hace el programa y las partes que lo conforman se explicará a continuación. Bienvenida es el nombre de la clase principal y por tanto, del archivo que contiene el código fuente (por convención de Java, el archivo donde se guarda el código fuente deberá tener el mismo nombre de la clase principal). Como ya se ha explicado antes, todos los programas en Java tienen un método principal de entrada conocido como main que a su vez contiene (encierra) un conjunto de sentencias que significan las instrucciones que deberá ejecutar el programa, este conjunto de instrucciones o bloque de instrucciones en Java se indican con la utilización de llaves { }. En el programa contenido en la clase bienvenida el bloque se compone sólo de tres instrucciones de impresión a pantalla.

Utilización de comentarios. Desde el surgimiento de los lenguajes de programación, junto con ellos, de igual manera nació la necesidad de hacer anotaciones sobre lo que hace tal instrucción, un método completo o inclusive el programa entero. Esta necesidad se resolvió con la introducción del soporte de comentarios (notas) dentro del mismo código fuente que se pueden emplear de distintas formas, por ejemplo recordar para qué se usó una variable específica, para documentar los valores de retorno de un método, entre muchos otros. Los comentarios que se utilizan en Java son del mismo estilo que los empleados en C/C++, así el compilador al comenzar a hacer su trabajo ignora por completo lo que esté entre estos comentarios, ya que no forman parte del código del programa, sólo son líneas de carácter informativo para el programador humano. Hay comentarios de una línea y de varias líneas, por ejemplo:

// Este es un comentario escrito en una sola línea y para hacer comentarios breves y concisos.

Como se puede observar el comentario comienza con un par de caracteres del tipo "/" y nótese que no se deja espacio alguna entre ellos, después de esto se puede comenzar a escribir el cuerpo del comentario de una sola línea. Cuando hay la necesidad de hacer observaciones muy amplias y que una línea no dé el espacio suficiente se sugiere utilizar comentarios de varias líneas, por ejemplo:

/*
Este es un comentario muy amplio y se puede hacer con la utilización de varias líneas sin perder la continuidad de la idea e indicándole al compilador que ignore por completo todas las líneas que lo comprenden.
*/

De igual forma que en los comentarios de una sola línea, los comentarios multilínea comienzan con el par de caracteres "/*" y terminan con el par de caracteres "*/".

Además en Java existe un tercer tipo de comentario que se conoce como "comentario de documentación", y la función que tienes es que le indica al programa generador de documentación que lo incluya a él e ignore de los demás, por ejemplo:

/** Comentario de documentación del programa */

Se debe notar que la forma de comenzar el comentario es ligeramente diferente al comentario multilínea, ya que ahora se incluye un asterisco más, para quedar como "/**". La forma de terminarlo es la misma.

Como sugerencia sobre los comentarios se debe hacer notar que, la necesidad de escribir comentarios es para la aclaración de ciertas partes del código que no estén muy claras, por lo tanto si se exagera en el número de comentarios se entenderá que el código no es claro y se deberá considerar la necesidad de reescribirlo para hacerlo comprensible.

Identificadores

Se entenderá al identificador como aquella palabra o nombre que el programador le asigna a sus propias variables, métodos e inclusive nombres de clases y tiene un ámbito de aplicación e injerencia sólo dentro del código fuente, escrito por el programador humano, porque una vez que pasa por el proceso de compilación, el compilador genera sus propios identificadores más acorde y más óptimos para su propia interpretación dentro del bytecode. Como regla general del lenguaje Java se debe definir un identificador antes de ser utilizado. Siguiendo las reglas que Java establece para la creación de identificadores, se deberá apegar a lo siguiente:

- Un identificador comienza por una letra o utilizando el carácter de guión bajo (_), inclusive utilizando el símbolo de pesos ($) aunque esto último no se recomienda ya que, como se explicó, el compilador lo utiliza de forma muy frecuente para formar sus propios identificadores.
- Se pueden utilizar a continuación números (0 al 9). Pero no dejar espacios en blanco, signo de interrogación (?) o símbolo de porcentaje (%).
- Un identificador puede contener la longitud que el programador asigne arbitrariamente o considere necesario para dejar perfectamente claro el propósito del identificador y qué tipo de dato alojará.
- En Java, al ser sensible al contexto, se distinguen los caracteres en mayúscula y minúscula. Así promedio, Promedio y PROMEDIO serán locaciones de memoria completamente diferentes.
- No se puede repetir el mismo identificador para dos variables o cualquier otra parte del código (nombres de métodos o clases por ejemplo), porque como se ha mencionado, los identificadores hacen precisamente eso, identifican de manera única a cada parte del código.
- Aunque la escritura de identificadores no está normalizada en Java, se puede apegar a las siguientes recomendaciones:
- Identificadores autoexplicativos, para guiar al usuario (de cualquier tipo) a entender la utilización de dicho identificador sin la necesidad de utilizar comentarios. Por ejemplo, para alojar el resultado del promedio de n números enteros se sugiere utilizar el identificador resultadoPromedio o resAvg.
- Si el identificador está compuesto por dos o más palabras, éstas se agrupan, la primera de ellas se escribe totalmente en minúscula y la primera letra de las

subsecuentes se escribe con mayúscula, por ejemplo: resultadoPromedioCalificaciones o longitudMedida.

- Como se mencionó se puede utilizar el símbolo de guión bajo (_) pero por convención sólo se emplea al nombrar constantes, ya que estas (igual por convención) deberán escribirse en letra mayúscula y dificulta la lectura y separación de la misma, por ejemplo: VALOR_GENERAL_ANUAL.

1.3.2. Convenciones de la programación

- Paquetes (Packages): El nombre de los paquetes deben ser sustantivos escritos con minúsculas. *package shipping.object*

- Clases (Classes): Los nombres de las clases deben ser sustantivos, mezclando palabras, con la primera letra de cada palabra en mayúscula (capitalizada). *class AccountBook*
- Interfaces (Interfaces): El nombre de la interfaz debe ser capitalizado como los nombres de las clases. *interface Account*

- Métodos (Methods): Los métodos deben ser nombrados con verbos, mezclando palabras, con la primera letra en minúscula. Dentro del nombre del método, la primera letra de cada palabra capitalizada. *balanceAccount()*

- Variables (Variables): Todas las variables deben ser combinaciones de palabras, con la primera letra en minúscula. Las palabras son separadas por las letras capitales. Moderar el uso de los guiones bajos, y no usar el signo de dólares ($) porque tiene un significado especial dentro de las clases. *currentCustomer*

- Las variables deben tener significados transparentes para exponer explícitamente su uso al lector casual. Evitar usar simples caracteres (i, j, k) como variables excepto para casos temporales como ciclos.

- Constantes (Constants): Las constantes deben nombrarse totalmente en mayúsculas, separando las palabras con guiones bajos. Las constantes tipo objeto pueden usar una mezcla de letras en mayúsculas y minúsculas. *HEAD_COUNT MAXIMUN_SIZE.*

- Estructuras de Control (Control Structures): Usar Llaves ({ }) alrededor de las sentencias, aun cuando sean sentencias sencillas, son parte de una estructura de control, tales como un if-else o un ciclo for (estas estructuras se explicarán a fondo en la siguiente unidad).

```
if ( condición ){
Sentencia1;
}
else
{
Sentencia2;
}
```

- Espaciado (Spacing): Colocar una sentencia por línea, y usar de dos a cuatros líneas vacías entre sentencias para hacer el código legible. El número de espacios puede depender mucho del estándar que se use.
- Comentarios (Comments): Utilizar comentarios para explicar los segmentos de código que no son obvios. Utilizar // para comentar una sola línea; para comentar extensiones más grandes de información encerrar entre los símbolos delimitadores /* */. Utilizar los símbolos /** */ para documentar los comentarios para proveer una entrada al javadoc y generar un HTML con el código:

```
//Un comentario de una sola línea
/*Comentarios que pueden abarcas más de
una línea*/
/**Un cometario para propósitos de documentación
*@see otra clase para más información
*/
```

Nota: la etiqueta @see es especial para javadoc para darle un efecto de “también ver” a un link que referencia una clase o un método.

1.3.3. Palabras reservadas

Se entiende como palabra reservada a un tipo especial de identificadores que sólo puede emplear el lenguaje Java, es decir su uso es exclusivo del lenguaje y el programador no

puede hacer uso de ellas para nombrar métodos, clases, variables o cualquier otra parte del código fuente, se presentan estas palabras reservadas en la siguiente tabla:

Abstract	Do	implements	protected	Throw
Boolean	Doublé	import	Public	Throws
Break	Else	instanceof	Rest	Transient
Byte	Extends	Int	return	True
Case	False	interface	Short	Try
Catch	Final	Long	Static	Void
Char	Finally	native	strictftp	Volatile
Class	Float	New	Super	While
const*	For	Null	switch	
Continue	goto*	package	synchronized	
Default	If	private	This	

Aunque la definición del lenguaje Java toma en cuenta todas estas palabras, en su implementación no se toman en cuenta todas, las que están marcadas con un asterisco aún no tienen utilización.

También existen palabras reservadas de empleo específico para nombres de métodos y éstas son:

Clone	Equals	finalize	getClass	hashCode
Notify	notifyAll	toString	Wait	

1.3.4. Estructura de una clase

Apegándose estrictamente a los lineamientos que genera la documentación de Java, una clase deberá contener las siguientes partes:

- Package: Es el apartado que contendrá la definición de directorios o jerarquía de acceso a donde estará alojada la clase.
- Import: Sentencia necesaria para poder “invocar” todos las paquetes necesarios (contenidos en otros packages) para que nuestra aplicación funcione. Por ejemplo, al tratar de utilizar una conexión hacia una base de datos, se necesitará importar los paquetes que proporcionan la funcionalidad necesaria. Algunos paquetes de Java se importan automáticamente, no es necesario hacer un import explícito, como la librería java.lang

- Class: Definición del nombre de clase. Se determina el nombre identificador que contendrá la clase.
- Variables de ámbito de clase: Se determinan las variables que se usarán en la clase para guardar datos y hacer operaciones.
- Constructor de la clase: El constructor es un tipo especial de método, que se invoca automáticamente al hacer una instancia del objeto. Sirve para inicializar los objetos, las variables o cualquier configuración inicial que necesite la clase para funcionar. Se puede sobrecargar.
- Métodos: Bloques de código que contienen las instrucciones y declaraciones que ejecutará el programa.

Ejemplo	Uso

`package javaapplication1;`
`import javax.swing.*;`
`public class JavaApplication1 {`
`public JavaApplication1(){}`
`public static void main(String[] args) {`
`//instrucciones`
`}`
`}`
	Paquete
	import si se requiere
	declaración de la clase
	constructor
	declaración del método
	cuerpo del método
	cierre del método
	cierre de la clase

1.3.5. Declaración de objetos y constructores

El elemento básico de la programación en Java es la clase. Una clase define métodos (comportamiento) y atributos (forma) de un objeto.

Esto significa que se mapea la clase hacia un objeto. Allan Kay, citado en Zukowski, (1997), describió las cinco principales características de Smalltalk, uno de los primeros lenguajes orientados a objetos y uno de los lenguajes en los cuales está basado Java:

- **Todo es un objeto**: Considere un objeto una variable especial, no solamente guarda datos, sino también se pueden hacer solicitudes a este objeto en sí. En teoría, cualquier elemento en el problema espacial (real) (edificios, servicios, automóviles, u otra entidad) puede ser representado como un objeto en un programa.

- **Un programa es un conjunto de objetos**, enviándose mensajes entre sí, para realizar una solicitud a un objeto es necesario enviarle un mensaje. Concretamente se puede pensar que un mensaje es una solicitud para llamar una función que pertenece a cierto objeto.
- **Cada objeto tiene su memoria**, conformada por otros objetos: en otras palabras, es posible generar un tipo de objeto nuevo agrupando objetos existentes. De esta manera se pueden armar estructuras complejas en un programa, escondidas detrás de la simplicidad de objetos.
- **Todo objeto tiene su tipo**: en este sentido tipo se refiere a clase, donde cada objeto es una instancia de una clase en cuestión. La característica más importante de una clase es el tipo de mensajes que pueden ser enviados a ella.
- **Todo objeto de un mismo tipo puede recibir los mismos mensajes**: esto implica que si un objeto es del tipo círculo, un objeto del tipo figura será capaz de recibir mensajes de círculo, puesto que un círculo es una figura. Este concepto forma parte medular de todo lenguaje orientado a objetos y será explorado en una sección específica de este curso. (Zulowski, 1997, p.1)

empleado	Nombre de la clase de la cual se instanciará el objeto
objetoE	Nombre del objeto
=	Asignación
New	Palabra new para hacer la separación de la memoria.
empleado()	Invocación al constructor
=new empleado();	Cierre de la instanciación

Quedando la instrucción como sigue:

```
empleado objetoE=new empleado();
```

Constructores

Un constructor es un método especial en Java, empleado para inicializar valores en instancias de objetos, a través de este tipo de métodos es posible generar diversos tipos de instancias para la clase en cuestión; la principal característica de este tipo de métodos es que llevan el mismo nombre de la clase.

Si la clase se llama empleado su declaración sería:

```
public class empleado { …… }
```

El constructor debería quedar como sigue

```
public empleado(){}
```

Y se usan para poder crear instancias de la clase (objetos).

Cierre de la unidad

Has concluido la primera unidad del curso. A lo largo de ésta te has introducido a la programación orientada a objetos y sus principales características como abstracción, polimorfismo, encapsulación y herencia; también has estudiado la diferencia entre la orientación a objetos y la programación estructurada. Posteriormente, te involucraste en las características del lenguaje Java, sus tipos de datos soportados, tipos de operadores y conversión de tipos de datos. Estos temas han servido para que vayas conociendo el ambiente de desarrollo de programas computacionales.

Al final de la unidad también se vio la estructura de un programa, palabras reservadas, estructura de una clase, declaración de objetos y constructores, cuyo propósito fue proporcionarte un panorama general de programar.

Es aconsejable que revises nuevamente la unidad en caso de que los temas que se acaban de mencionar no te sean familiares o no los recuerdes, de no ser este tu caso, ya estás preparado(a) para seguir con la unidad dos, en donde continuarás con la revisión de los métodos y las estructuras de control. Todo ello con el fin de que obtengas el conocimiento necesario para comenzar a realizar pequeños programas computacionales al final de la cuarta y última unidad de la asignatura Programación orientada a objetos I.

Para saber más

Es importante que instales un IDE en tu computadora personal para que pases todos los ejemplos de código y veas cómo funcionan, de esta manera podrás analizar el funcionamiento de los códigos presentados.

*__Nota__: se recomienda que instales NetBeans 7.0, como IDE, por su facilidad de uso, este puede ser descargado gratuitamente de la siguiente liga: http://netbeans.org/downloads/

Fuentes de consulta

Bibliografía básica

- Devis, R. (1993). *C++/OOP: un enfoque práctico*. Madrid: Paraninfo. Recuperado de http://www.a4devis.com/articulos/libros/Programaci%C3%B3n%20Orientada-a-Objetos%20en%20C++.pdf
- Joyanes, L. (2001). Programación orientada a objetos versus programación estructurada: C++ y algoritmos. En *OLC - Programación en C++*. España: McGrawHill Interamericana. Recuperado de http://www.mcgraw-hill.es/bcv/guide/capitulo/8448146433.pdf
- Joyanes, L., y Fernández, M. (2001). *Java 2: manual de programación*. España: McGrawHill Interamericana.
- Zukowski, J. (1997). *The Java AWT Reference (Java Series), version 1.2.* O'Reilly & Associates. Recuperado de http://oreilly.com/openbook/javawt/book/index.html

Bibliografía complementaria

- Flanagan, D. (2005). *Java in a NutShell*. USA: O'Reilly & Associates.