DOXA EDITION

Autor:
ELVIS PACHACAMA C.

2023

Dominando Java I:

Aprende los pilares del desarrollo de software con lenguaje Java

Primera Edición

ITQ
WWW.ITQ.EDU.EC
INVESTIGACIÓN

ISBN: 978-9942-7156-0-9

INSTITUTO SUPERIOR TECNOLÓGICO QUITO
Excelencia en Educación Superior

DOMINANDO JAVA I:
APRENDE LOS PILARES DEL DESARROLLO DE SOFTWARE CON EL LENGUAJE JAVA

AUTOR:

ELVIS PACHACAMA C.

PRIMERA EDICIÓN

2023

TRABAJO EN EDICIÓN:

DOXA EDITION

EDITOR INTERNO: GINO CORDERO P.

EDITOR EXTERNO: DIEGO BASTIDAS L.

ISBN: 978-9942-7156-0-9

9 789942 715609

QUITO - ECUADOR

DEDICATORIA

Esta obra está dedicada a los dos pilares de mi vida, Grace Amparo Cabezas Salazar y José Ramón Pachacama Tipán, que sin ellos no hubiera podido lograr todo lo que tengo.

Gracias por todos sus esfuerzos lágrimas desvelos por verme triunfar.
Y para ti mi Keylita que estas en el cielo sé que me vas a cuidar y guiar cada paso que dé.

F. Elvis

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a todas aquellas personas que hicieron posible la creación de mi primer libro.

En primer lugar, quiero agradecer a mi familia por su amor, apoyo y paciencia durante todo el proceso. Gracias por haberme brindado el espacio y el tiempo necesario para poder concentrarme en esta tarea y por haber sido mi fuente de inspiración en cada página.

También quiero agradecer a mis amigos y colegas por su constante motivación, por haberme brindado retroalimentación y sugerencias valiosas durante la escritura de este libro. Gracias por su amistad y por haber compartido conmigo momentos de alegría y distracción.

Un agradecimiento especial a mi editor, por su profesionalismo, dedicación y paciencia en la revisión y edición de este libro. Gracias por haberme guiado en cada paso del proceso de publicación y por haber hecho posible la materialización de este proyecto.

Finalmente, quiero agradecer a mis lectores por su interés en mi trabajo y por brindarme la oportunidad de compartir mis ideas, pensamientos y pasión por la educación. Espero que este libro les brinde una experiencia única y significativa

F. Elvis

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

SOBRE EL AUTOR

Elvis Pachacama Cabezas es un educador y profesional con título de tercer nivel en Ingeniería en Tecnologías de la Información obtenido en la Universidad Estatal de Sumy en Ucrania (SUMDU) y esta es su primera publicación. Fue representante de los estudiantes ecuatorianos en el Consejo Estudiantil de la universidad, graduado con honores con reconocimiento en su título universitario. Elvis es docente en el Instituto Superior Tecnológico Quito, donde imparte catedra en la Carrera de Desarrollo de Software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

CONTENIDO

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Introducción

Bienvenido al apasionante mundo de la programación en Java. Este libro está diseñado para ayudarlo a dar sus primeros pasos en el aprendizaje del lenguaje de programación Java y proporciona una base sólida para convertirse en un programador competente y versátil.

Java es un lenguaje de programación ampliamente utilizado y reconocido en la industria del desarrollo de software. Su popularidad se debe a su enfoque en la portabilidad, la flexibilidad y la facilidad de uso, lo que la convierte en la opción preferida para desarrollar de aplicaciones empresariales, aplicaciones móviles, sistemas integrados y muchas otras soluciones tecnológicas.

En este libro, nos sumergiremos en el mundo de Java desde cero, por lo que no se requiere experiencia previa en programación. Empezaremos con los conceptos fundamentales y poco a poco avanzaremos hacia temas más avanzados. El objetivo es que, al final de este curso, pueda crear programas funcionales y comprender cómo funciona Java.

¿Qué encontraras en este libro?

Introducción a Java. Este capítulo presenta técnicas de programación a programadores novatos y recuerda a los programadores experimentados estas técnicas básicas, y cubre técnicas utilizadas para resolver problemas informáticos.

Las etapas del método clásico de desarrollo de programas incluyen: análisis de problemas, desarrollo de algoritmos, codificación o implementación de algoritmos en un lenguaje de programación de alto nivel, compilación, ejecución del programa original, verificación y prueba del programa; seguido del mantenimiento y documentación del programa.

Aunque este libro se ocupa principalmente del desarrollo práctico de programas, técnicas y métodos, se introducen brevemente las fases de programación clásicas, como el análisis de requisitos y la especificación del dominio del problema a resolver; para ello cabe mencionar que sí, los dos modelos de programación más utilizados en el ámbito educativo y profesional son: la programación estructurada y la programación orientada a objetos; porque Java es un lenguaje completamente orientado a objetos.

Metodología de programación, creación y desarrollo de programas en Java.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Este capítulo presenta técnicas de programación a programadores novatos y recuerda a los programadores experimentados estas técnicas básicas, y cubre técnicas utilizadas para resolver problemas informáticos.

Las etapas del método clásico de desarrollo de programas incluyen: análisis de problemas, desarrollo de algoritmos, codificación o implementación de algoritmos en un lenguaje de programación de alto nivel, compilación, ejecución del programa original, verificación y prueba del programa; seguido del mantenimiento y documentación del programa.

Aunque este libro se ocupa principalmente del desarrollo práctico de programas, técnicas y métodos, se introducen brevemente las fases de programación clásicas, como el análisis de requisitos y la especificación del dominio del problema a resolver; para ello cabe mencionar. Es importante señalar que los dos modelos de programación más utilizados en el ámbito educativo y profesional son: la programación estructurada y la programación orientada a objetos; Debido a que Java es un lenguaje totalmente orientado a objetos, este patrón se utiliza a lo largo del libro y sus aplicaciones son universales y están orientadas a Internet y a la Web.

Elementos básicos de Java. Hemos visto cómo crear nuestros propios programas, ahora analizaremos los conceptos básicos de Java.

Dado que este capítulo es importante en el desarrollo de aplicaciones, este capítulo cubre los conceptos teóricos y prácticos relacionados con la estructura del programa. Cubierto en el capítulo anterior, incluidos los siguientes temas:

- Estructura general de un programa en Java.
- Creación del programa.
- Elementos básicos que lo componen.
- Tipos de datos en Java y como se declaran.
- Concepto y declaración de variables.
- Operaciones básicas de entrada y salida.

Operadores y expresiones. Un programa escrito secuencialmente ejecutará declaraciones una tras otra; del primero al último, cada declaración se ejecutará solo una vez; El modo secuencial es adecuado para resolver problemas simples.

Pero para resolver problemas generales, es necesario poder comprobar el contenido de la notificación, siempre se debe hacer.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Una estructura o construcción de control determina la secuencia de ejecución o flujo de declaraciones; se divide en tres categorías según el flujo de ejecución: secuencia, selección y repetición.

Este capítulo analiza las sentencias if y switch, la selectividad o las construcciones condicionales que controlan si una sentencia o lista de sentencias se ejecuta en función de si se cumple una condición; Para soportar estas construcciones, Java tiene el tipo booleano.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Capítulo 1

Introducción a Java

1.1. Historia y Características de Java

Java[1] fue desarrollado por un equipo de ingenieros de Sun Microsystems a principios de la década de los 90, el proyecto fue liderado por James Gosling. Inicialmente, el proyecto se llamó "Green Project" y tenía como objetivo desarrollar una plataforma de software para dispositivos electrónicos de consumo. Sim embargo, pronto se dieron cuenta que las limitaciones de los dispositivos de la época no permitirían ejecutar aplicaciones complejas en ellos.

En 1995, Sun Microsystems lanzó oficialmente Java como una plataforma de desarrollo de software para aplicaciones empresariales y de escritorio. Su lema principal era" Write once, run anywhere" (Escribe una vez, ejecuta en cualquier lugar), lo que significa que el código Java puede ser compilado en u n formato llamado "bytecode" y ejecutado en cualquier máquina virtual Java (JMV), independientemente del sistema operativo subyacente.

Java rápidamente ganó popularidad debido a su enfoque en la portabilidad, la seguridad y su capacidad para ejecutar aplicaciones en múltiples plataformas sin necesidad de reescribir el código fuente. En 2009, Sun Microsystems fue adquirida por Oracle Corporation, quien se convirtió en el principal patrocinador y desarrollador de Java.

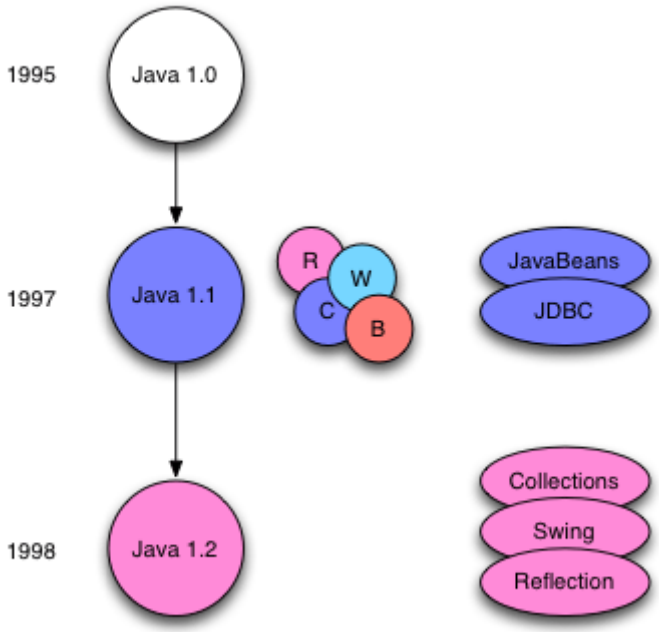
A lo largo de los años, Java ha experimentado varias versiones principales y actualizaciones, cada una introduciendo nuevas características y mejoras. A continuación, se presenta un resumen de la historia de Java y sus versiones principales.

[1] La fuente cuenta con una extensa documentación sobre la historia de Java y la evolución en el tiempo https://www.netec.com/post/historia-y-curiosidades-de-java

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

1.1.1. Evolución de Java

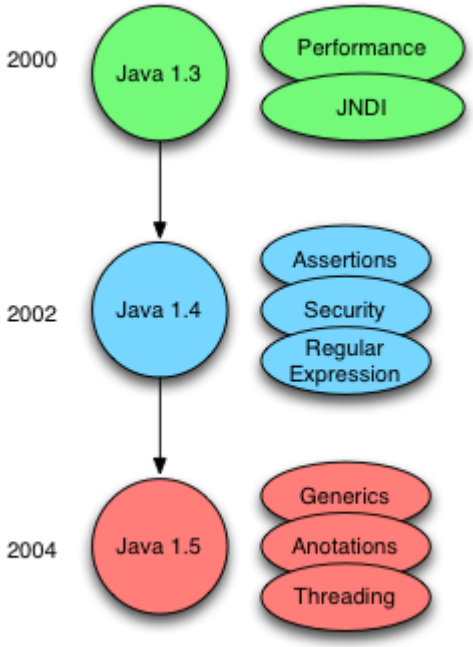
Figura 1
Versión Java (1-2)



Nota. La figura representa los cambios de java desde la versión 1.0 hasta la versión 1.2. Tomada de https://www.arquitecturajava.com/las-versiones-de-java/

- **JDK 1.0(enero de 1996):** Fue la primera versión oficial de Java, Introdujo el lenguaje de programación Java, la máquina virtual de Java (JMV) y las bibliotecas básicas de clases.
- **JDK 1.1(febrero de 1997):** Esta versión agregó nuevas bibliotecas y funcionalidades como, soporte para aplicaciones de red y acceso a base de datos mediante JDBC (Java Database Connectivity).
- **J2SE 1.2(diciembre de 1998):** Se introdujo la nueva marca "Java2"

Figura 2
Versión de Java (1.3-1.5)

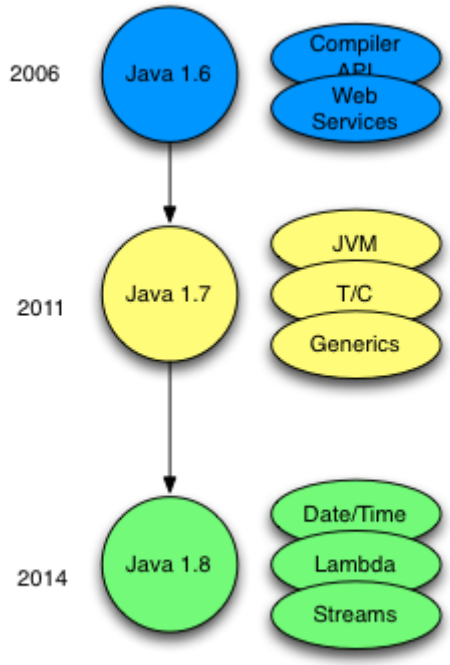


Nota. La figura representa los cambios de java desde la versión 1.3 hasta la versión 1.5. Tomada de https://www.arquitecturajava.com/las-versiones-de-java/

- **Versión 1.3:** Avances pequeño en cuanto a APIs, se añade soporte JNDI. Sin embargo, el avance en cuanto a la arquitectura de la maquina virtual es importante ya que aparece la máquina HotSpot con compilación JIT (**Just-in Time**)
- **Versión 1.4:** Se produce un salto importante en cuanto a nuevas Apis. Se incorpora un fuerte soporte XML, Expresiones Regulares, criptografía, etc.
- **Versión 1.5:** También denominada Java 5 se producen dos altos importantes a nivel del core del lenguaje. Por una parte, la inclusión de tipos genéricos que no tenía el mundo de las colecciones. Por otro lado, la inclusión del **concepto de metadatos con el uso de anotaciones.** Se amplía el soporte de APIs orientadas a la programación concurrente.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 3
Versión de Java (1.6- 1.8)



Nota. La figura representa los cambios de java desde la versión 1.6 hasta la versión 1.8. Tomada de https://www.arquitecturajava.com/las-versiones-de-java/

- **Versión 1.6:** Esta versión contiene avances muy puntuales con la inclusión de un API de compilación "on-the-fly" que permitirá gestionar servicios web de forma cómoda.
- **Versión 1.7:** Otra versión cuyos cambios a nivel del lenguaje son imitados. Se produce una mejora de la máquina virtual incluyendo nuevos recolectores de basura.
- **Versión 1.8:** Llega Java 8 el gran salto en cuanto al lenguaje se refiere. Se abren las puertas a la programación funcional con el uso **de expresiones Lambda y Streams.** Se realiza una revisión de APIS y se actualiza de forma importante la gestión de fechas.
- **Versión 1.9:** En este reléase Java incluye el proyecto Jigsaw que permite modularizar el JDK. Y la distribución de packages. Se trata de un proyecto clave para el futuro enfoque de MicroServicios.
- **Versión 1.10:** Quizás la mejora más significativa es el añadido de la palabra **var** al lenguaje simplificado la inferencia de tipos cuando programamos de tal forma que a partir de ese momento es válido el uso de la siguiente estructura:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
var lista= new ArrayList<Persona>();
```

Aparte de esto mejoraron temas de recolector de basura y programación concurrente.

- **Versión 1.12:** Añade la suite de MicrosoftBenchMark que permite al desarrollador realizar pruebas de rendimiento dentro del propio JDK.
- **Versión 1.15:** Fue lanzada el 15 de septiembre de 2020. Esta versión continúa la evolución de la plataforma Java y presenta diversas características y mejoras que mejoran la productividad de los desarrolladores y la eficacia del lenguaje.
- **Versión 1.17:** Fue lanzada el 14 de septiembre de 2021 y es una versión de soporte a largo plazo.
- **Versión 20:** La última actualización de Java, JDK 20, presenta versiones preliminares o de incubación de siete capacidades nuevas, incluidos subprocesos virtuales y concurrencia estructurada.

  Las siete funciones marcadas oficialmente para el lanzamiento, todas las cuales están en etapa de incubación o de vista previa, también incluye una API de vector, valores de ámbito, patrones de registros, coincidencia de patrones para declaraciones y expresiones de cambio, y una API de función y memoria externa.

1.2. Configuración del Entorno de Desarrollo

Java requiere algunas configuraciones para poder usarlo en nuestro equipo. Tenemos que tener en cuenta que puede haber

La configuración del entorno de Java implica instalar y configurar el Java Development Kit (JDK) y, opcionalmente, un entorno de desarrollo integrado (IDE) para facilitar el desarrollo de aplicaciones en Java. A continuación, se presenta una guía para configurar el entorno de Java:

a. Instalación del JDK

Visita el sitio web oficial del Oracle https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.exe (sha256) y descarga la versión más reciente del JDK compatible con tu sistema operativo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

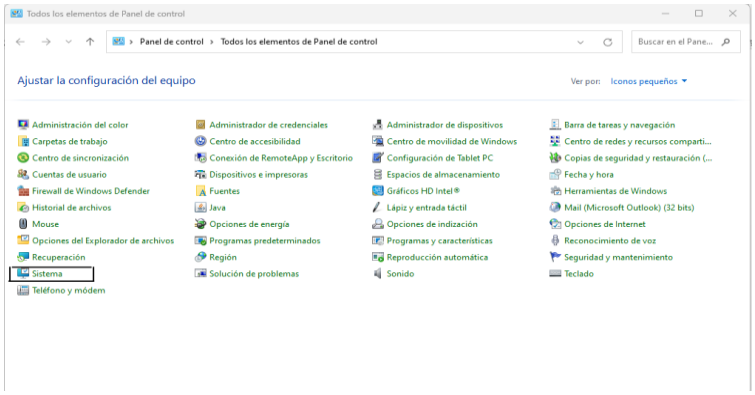
Figura 4
Instalación de JDK

Nota. La figura representa el inicio de la instalación del JDK. Captura de pantalla

b. Configuración de las variables de entorno:

- Abre las variables de entorno del sistema en tu SO. En Windows, puedes hacerlo a través del Panel de control ->Sistema-> Configuración avanzada del sistema->Variables de entorno. En Linux y macOS, puedes editar el archivo. bashrco o .bash_profile en tu directorio de inicio.

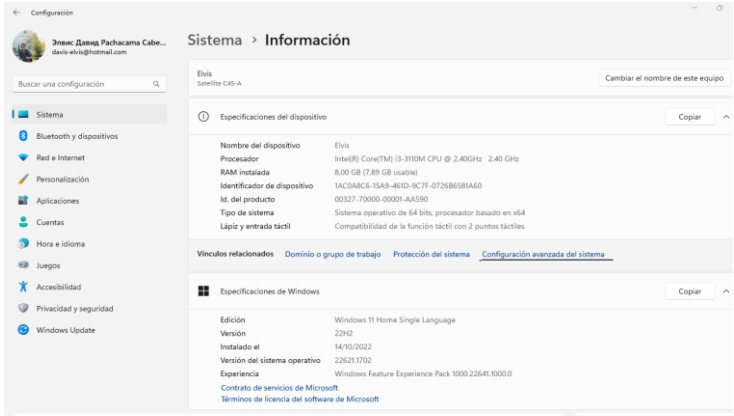
Figura 5
Panel de control-Sistema



Nota. El figura representa el Panel de control de Windows para la configuración de las variables de entorno. Captura de pantalla

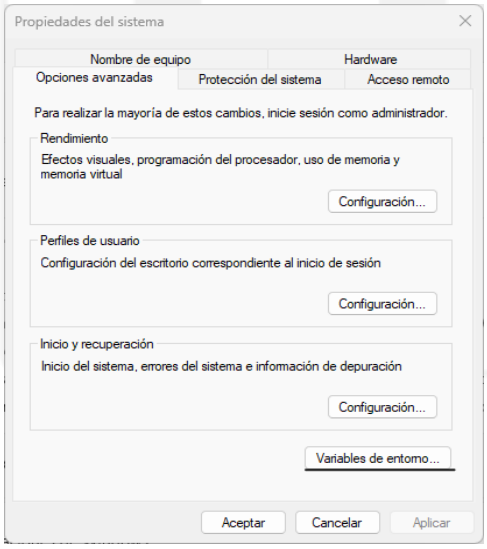
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 6
Configuración avanzada



Nota. La figura representa la configuración de variables de entorno, mediante configuración avanzada. Captura de pantalla

Figura 7
Variables de entorno



Nota. La figura representa donde se debe configurar las variables de entorno en el sistema operativo Windows. Captura de pantalla

- Agregamos la ruta de instalación del JDK a la variable de entorno “PATH”. Esto permite que el sistema encuentre los comando y herramientas del JDK. Por ejemplo, en Windows, podrías agregar.
- Instalación de un IDE. Hay varios IDEs populares para el desarrollo de Java, como Eclipse, IntelliJIDEA y NetBeans. Elije el IDE que prefieras y descárgalo desde su sitio web oficial.

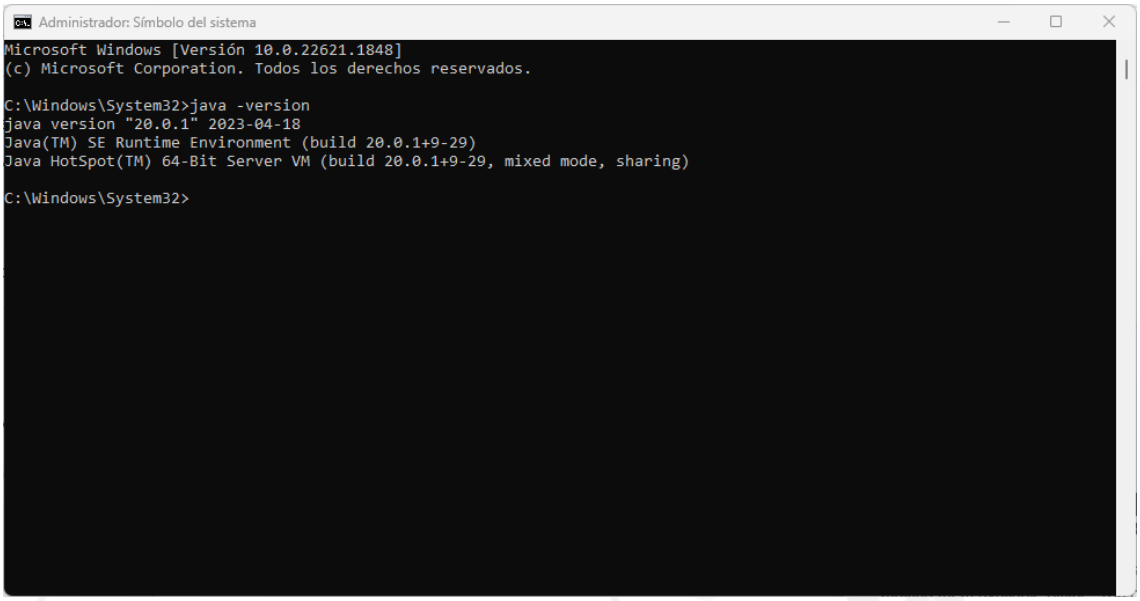
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Sigue las instrucciones del instalador del IDE para completar la instalación en tu sistema.

Una vez instalado el IDE, puedes configurar la ruta JDK en la configuración del IDE para asegurarte que estas utilizando la versión correcta del JDK.

Después de completar estos pasos, tu entorno de Java debería estar configurado y listo para desarrollar aplicaciones en Java Puedes verificar la configuración ejecutando el comando "java -version" en la línea de comandos, que debería mostrar la versión del JDK instalado.

Figura 8
Versión de Java



Nota. La figura representa la versión de Java que tenemos instalado en nuestro entorno de trabajo. Captura de pantalla

1.3. La Máquina Virtual Java, Estructura de un Programa en Java

Vamos a crear nuestro primer programa, que nos servirá para entender y comprobar si hemos instalado y configurado correctamente Java antes vamos a reparar algunas definiciones importantes que nos permitirá comprender lo que vamos haciendo.

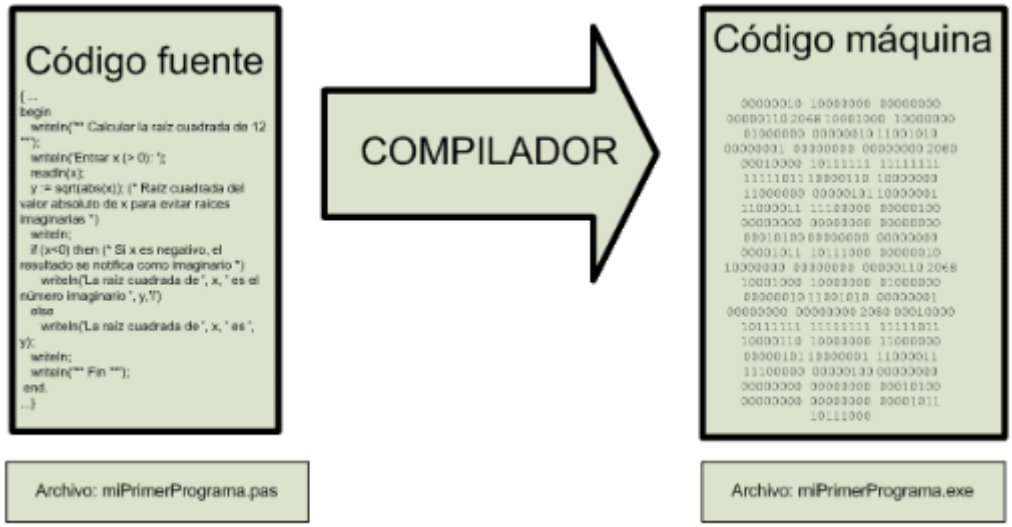
- **Compilación.** "Compilar" significa traducir el código escrito en "Lenguaje entendible por humanos", por ejemplo, Java, C, Pascal, etc. A un código que entienda la computadora o código de máquina, este lenguaje no es entendido por nosotros. Se hace esto porque para nosotros nos seria imposible trabajar directamente con el lenguaje de máquina. Es por esta razón que en este libro vamos a utilizar el lenguaje de programación Java y luego emplearemos un traductor (compilador). La creación de programas en muchos lenguajes de

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

programación se base en el siguiente proceso: escribir código fuentes, este código fuente se compila y se obtiene un programa ejecutable.

El compilador se encarga de evitar que un programa se pueda traducir su código fuente con errores y de hacer otras verificaciones previas, de modo que el código de máquina va a tener ciertas garantías de que cumplen cuando menos con los estándares de sintaxis obligatorios de un lenguaje.

Figura 9
Compilación de Programa en Pascal



Nota: Uno de los grandes problemas al compilar este programa es que si lo desarrollaste en un Sistema Operativo de Windows solo se va a ejecutar en Windows y no lo podrás hacer en Unix, Linux u otro Sistema Operativo.

Una aplicación creada en Java no corresponde a la figura 9. Esta es una de las características novedosas que tenía Java con relación a otros lenguajes de programación cuando se creó la primera versión. Lo novedoso de Java fue que se hizo independiente del hardware y del Sistema Operativo en el que se ejecutaba.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen del Capítulo 1

En el capítulo de introducción a Java, se abordan los fundamentos esenciales de este lenguaje de programación. Java es un lenguaje versátil y orientado a objetos que se utiliza para desarrollar una amplia variedad de aplicaciones, desde aplicaciones web hasta aplicaciones móviles y sistemas embebidos.

El capítulo comienza explicando una breve historia del lenguaje de programación Java desde su primera versión hasta la versión final que hoy en día se está utilizando, como fue evolucionando cada una de las versiones.

El capítulo también se centra como configurar el entorno de desarrollo de Java, que incluye la instalación del Kit de Desarrollo de Java (JDK) y la configuración de las variables de entorno

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Capítulo 2

Metodología de Programación, Creación y Desarrollo de Programas en Java

2.1. Resolución de Problemas con Java

En Java el proceso de resolución de problemas utilizando una computadora implica la escritura de programas y su posterior ejecución; por lo tanto, la programación es un proceso de resolución de problemas y preguntas. Existen diferentes técnicas para solucionar estos problemas, aunque el diseño y las propuestas arquitectónicas de software son creativas y se pueden considerar diferentes fases durante la programación. Las fases de resolución de problemas y sus características más importantes son:

- **Análisis del problema.** En el análisis del problema se examina todas las especificaciones de los requisitos entregados por el cliente, respecto al programa.
- **Diseño del algoritmo.** Una vez que se haya analizado el problema a resolver, se diseña una solución de un conjunto de instrucciones llamado algoritmo que o resuelva.
- **Codificación.** Para la solución del algoritmo se escribe en la sintaxis en cualquier lenguaje de programación de alto nivel, en este caso Java, el cual se obtendrá un código fuente que después será compilado.
- **Compilación y ejecución.** Es este punto el programa se empaqueta, en el caso de Java se interpreta y se ejecuta.
- **Verificación y depuración.** Una vez ejecutado nuestro programa, se verifica rigurosamente su sintaxis y se elimina todos los errores que aparezcan, a estos errores se los llama *bugs*.
- **Mantenimiento.** En este punto nuestro programa se actualiza y se modifica cada vez que sea necesario para que cumplan todas las necesidades y requerimientos del usuario, en esta fase se utilizan y mejoran los algoritmos realizando los cambios que sean necesarios.
- **Documentación.** En esta última fase del desarrollo de un programa vamos a documentar, todas las fases del ciclo de vida del software, esencialmente el análisis del problema, diseño y codificación; junto con los manuales de usuarios y manuales técnicos, así como las normas para el mantenimiento del software.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

2.1.1. Análisis del Problema

En esta fase se requiere especificar el problema y definir claramente las tareas que el programa debe realizar y el resultado o solución que se necesita; en esta fase se divide en diferentes etapas.

- Entender el problema lo más claro posible.
- Comprender y enumerar los requerimientos o los requisitos del problema. Es necesario aclarar si nuestro software requiere interactuar con el usuario para la lectura de datos de entrada y también para poder especificar el formato de salida o los resultados.
- Cuando hablamos de especificar los datos supone que debemos representarlos en su formato correspondiente.
- Cuando nuestro programa realiza una salida de datos, se debe especificar cómo generar y dar formato a los resultados de salida.

Cuando el problema es complejo es necesario dividirlo o descomponerlo en subproblemas o módulos, una vez dividido el problema vamos aplicar los pasos anteriores para poder analizar individualmente los requerimientos necesarios, así como la posible relación o conexión entre cada uno de los módulos. El análisis del problema requiere una definición clara, considerando exactamente lo que hará el programa y la solución deseada; dentro de esta fase se realizaran algunas preguntas a tener en cuenta.

- ¿Qué datos de entrada se requiere?
- ¿Cuál es la información de salida que se requiere?

2.1.2. Diseño del Algoritmo

Después de haber analizado nuestro problema, la descripción y las especificaciones necesarias del mismo, el paso a seguir es diseñar un algoritmo que nos permita resolver dicho problema. Para esto nuestra computadora necesita que se le indique las tareas o acciones que se van a ejecutar en un orden sucesivo. Un algoritmo[2] es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

[2] La fuente cuenta con una extensa documentación sobre los conceptos, técnicas y métodos de diseño y construcción de algoritmos y programas: Joyanes L. Fundamentos de programación Algoritmos estructuras de datos y objetos 4a edición Madrid McGraw-Hill 2008

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Los pasos sucesivos que indican las acciones o instrucciones a ejecutar por la máquina constituyen el algoritmo; para completarlo se requiere el diseño previo del mismo, lo cual es independiente, lo cual es independiente tanto del lenguaje de programación en que se expresará como de la computadora que lo ejecutará. En la resolución del problema, el algoritmo se puede expresar en un lenguaje de programación diferente y ejecutarse en una computadora distinta, pero será siempre el mismo; por ejemplo: en una analogía de la vida diaria, una receta de cocina se puede expresar en español, ingles o francés, pero independientemente del lenguaje que hable el cocinero, los pasos de la receta serán los mismos.

La especificación del orden del algoritmo en donde se realizan las instrucciones o acciones del programa se llama control del programa, este control se realiza con ordenes secuenciales o repetitivas.

Estos controles se estudiarán en capítulos más adelante. A estas instrucciones que realizamos en lenguajes de alto nivel también se los llama sentencias.

En la etapa del análisis del proceso de programación se va a determinar lo que el programa hará, en la siguiente etapa de diseño se definirá como se va a realizar la tarea solicitada.

Uno de los métodos más fáciles para resolver un problema complejo es dividiendo, eso quiere decir que nosotros vamos a subdividir nuestro problema es subproblemas y luego fraccionando estos subproblemas en otros de nivel más bajo hasta que se pueda implementar una solución, este método es conocido como método de diseño descendente o modular.

Cualquier *software* bien diseñado consta de un programa principal, este programa viene siendo el módulo de más alto nivel, el cual llama a otros módulos o subprogramas de nivel más bajo, que a su vez estos pueden llamar a otros subprogramas. Estos módulos pueden diseñarse, codificarse, comprobarse y depurarse independientemente, incluso pueden ser desarrollados por distintos programadores para luego ser acoplados en un solo software.

Como ya sabemos el proceso para el desarrollo de software hasta que se concluya son los siguientes.

1. Programar un módulo.
2. Comprobarlo
3. Depurarlo, si es necesario
4. Integrarlos con los módulos anteriores.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

El proceso que va a convertir el análisis del problema en un diseño modular con refinamientos sucesivos que nos va a permitir una futura traducción a un lenguaje de programación de alto nivel se llama diseño del algoritmo, este diseño del algoritmo es muy aparte del lenguaje de programación en el que se va a codificar el programa.

El último de los pasos es el diseño del algoritmo en el cual se debe comprobar y verificar la exactitud de la solución del problema, esto quiere decir que vamos a obtener resultados exactos en un tiempo finito.

Los algoritmos se pueden representar de distintas formas, gráficamente por medio de fórmulas, diagramas de flujo N-S y seudocódigos, esta última técnica es la más utilizada en el mundo de la programación moderna y es la que más se recomienda para trabajar en Java. Por ejemplo, algoritmos en el cual nosotros podemos realizar tareas, “ir al estadio a ver un partido de futbol “Nacional-Barcelona””.

Ejemplo 2.1

1. Inicio
2. Ver la programación del campeonato de futbol en la página oficial de la LigaPro
3. **Si** no juega “Nacional-Barcelona” **entonces**
   - **3.1.** Decidir otra actividad
   - **3.2.** Bifurcar al paso 7
4. Si hay fila entonces
   - **4.1.** Formarse
   - **4.2.** **Mientras** haya personas delante **hacer**
     - **4.2.1.** Avanzar en la fila
     - **Fin de mientras**
   - **Fin de si**
5. **Si** hay localidades **entonces**
   - **5.1.** Compramos una entrada
   - **5.2.** Ingresamos a nuestra localidad
   - **5.3.** Localizamos nuestro asiento

Mientras se juega el partido hacer

- **5.3.1.** Ver y alentar al Barcelona

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Fin_mientras

Abandonar el estadio

6. Volver a casa
7. Fin

Como podemos ver en el algoritmo anterior tenemos algunos aspectos a considerar. Primero tenemos algunas palabras reservadas que están escritas en negritas como por ejemplo (entonces, mientras, si_no). Estas palabras son fundamentales para el desarrollo de nuestro problema, ya que estas van a estar encargadas en la toma de decisiones de los procesos de automatizaciones de nuestro software, dentro del algoritmo también tenemos conceptos importantes de selección como estas (si-entonces, si_no, if-then-else), también tenemos palabra que me van a permitir repetir o realizar ciclos a secciones de nuestro algoritmo estas palabras pueden son (mientras-hacer, hacer-mientras o repetir-hasta o iterar- fin de iterar), como gran parte de los lenguajes de programación son desarrollados en ingles las palabras reservadas que vamos a utilizar son while-do y do-while que se encuentra en la mayoría de los algoritmos, especialmente los que se procesan datos. Cuando usamos bucles de decisión nos va a permitir seleccionar distintas alternativas de procesos o acciones a seguir o indicar la repetición una y otra vez las operaciones básicas.

2.1.3. Codificación

Una vez terminado las dos primeras fases empezamos con la codificación del algoritmo desarrollado anteriormente, para esto elegimos el lenguaje de programación que nosotros deseemos en este caso vamos a utilizar el lenguaje de programación Java. Como ya sabemos el algoritmo es independiente del lenguaje de programación utilizado, el código puede escribirse de una manera más fácil.

Como dijimos anteriormente para codificar nuestros algoritmos lo que debemos hacer es sustituir las palabras reservadas en español por su equivalente en inglés, recordemos que los lenguajes de programación son desarrollados en inglés por ese motivo vamos a reemplazar nuestras palabras reservadas, y todas las operaciones e instrucciones indicadas en nuestros algoritmos de lenguaje natural a lenguaje de programación de alto nivel, aquí vamos a seguir todas las reglas de sintaxis de este.

Al terminar de diseñar nuestro algoritmo y verificar que funcione correctamente, se procede a traducir esas instrucciones a un lenguaje de alto nivel como ya lo habíamos mencionado a Java en algunas de sus versiones actuales como ya lo vimos en el capítulo anterior, preferiblemente versiones actuales. Una que hemos escrito el código fuente lo

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

que tenemos que hacer el pasarlo a un computador, mediante un editor de texto siguiendo las reglas de sintaxis de Java, uno de los objetivos de este libro es que el lector nunca omita este pequeño detalle.

En el caso de Java, el código fuente se guarda en un archivo de texto el cual tiene una extensión .java, por ejemplo nombreDelPrograma.java después de haber generado el código fuente el siguiente paso es la compilación del código fuente. Si en el código fuente existe algún fallo, el compilador envía un mensaje de error, el programador debe encontrar y solucionar esos errores para luego enviar nuevamente al compilador estos pasos se deben realizar las veces que sean necesarias hasta que el compilador no envíe ningún error. Una vez que el compilador no envié ningún mensaje de error el compilador genera el código de máquina o el bytecode en el caso de Java.

2.1.4. Compilación-Interpretación de un Programa en Java

Una vez escrito el código fuente este código se debe traducir a un lenguaje de máquina, este proceso lo realiza el compilador y el sistema operativo. La ejecución del programa lo hace la Máquina Virtual de Java una vez que el compilador de paso al código fuente eso quiere decir que ya no tenga errores de sintaxis.

La etapa siguiente a la compilación es la ejecución de nuestro programa, en esta etapa se verifica la compilación exitosa eso quiere decir que el programa no tenga ningún error de sintaxis, esto no quiere decir que el programa funcione correctamente. Puede ocurrir que el programa interrumpa su ejecución y termina de forma inesperada por la existencia de errores lógicos, como por ejemplo una división de un número entre 0 o una mala identificación de una variable o un dato. Cuando sucede este tipo de errores lo que toca hacer es revisar todo el código fuente el algoritmo y en muchos casos es necesario revisar el análisis y las especificaciones del problema, esto se debe a que un análisis incorrecto o un mal diseño de especificaciones puede producir un mal algoritmo.

Este proceso se repite una y otra vez hasta que nuestro programa no produzca ningún error, después de la compilación se obtiene un programa todavía no ejecutable directamente llamado programa objeto. Suponiendo que no existan errores de compilación en nuestro programa fuente, se debe indicar al sistema operativo que se ejecute la fase de ensamblaje o vinculación, cargando el programa objeto con las bibliotecas de compilación, esta operación se realiza con un cargador.

2.1.5. Verificación y Depuración de un Programa en Java

La verificación o depuración de un programa es el proceso de la ejecución con una amplia variedad de datos de entrada llamados de prueba que determinaran si el software tiene

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

errores bugs, para verificar la funcionalidad de nuestro software se debe utilizar una gran variedad de datos de distintos tipos para dichas pruebas y de esta forma se comprueba los límites y aspectos especiales de nuestro programa.

Los errores que se pueden encontrar en la verificación del software son:

1. Los errores de compilación son problemas detectados por el compilador de un programa informático al intentar traducir el código fuente escrito por un programador a un lenguaje que la computadora puede entender. Estos errores representan discrepancias o problemas en la sintaxis o estructura del código, impidiendo que el programa se compile correctamente y, por lo tanto, se ejecute. En otras palabras, los errores de compilación son fallos que deben corregirse antes de que el programa pueda funcionar adecuadamente, ya que el compilador no puede crear un programa ejecutable a partir del código fuente que contengan estos errores.
2. Los errores de ejecución se refieren a problemas que surgen cuando un programa informático está en proceso de ejecución o funcionamiento. Estos errores no se detectan durante la compilación del programa, sino que ocurren mientras el programa se está ejecutando. Como por ejemplo una división entre 0, estos resultados de situaciones inesperadas o condiciones inadecuadas que el programa no está preparado para manejar.
3. Los errores lógicos, también conocidos como errores de lógica, son fallos de un programa informático que no causan problemas al momento de su compilación o de ejecución, pero lo que si afecta negativamente el funcionamiento correcto del software debido a una lógica incorrecta o mal implementada en el código. Estos errores no suelen generar mensajes de error o alertas, por lo que pueden ser difíciles de detectar y suelen requerir un análisis cuidadoso del código para ser identificados y corregidos, si los errores persisten toca revisar el algoritmo y si persisten toca revisar el análisis del problema.

2.1.6. Documentación y Mantenimiento

La documentación se refiere al proceso de crear registros detallados y descripciones escritas que explican el funcionamiento de un sistema, programa o proceso. La documentación se utiliza para facilitar la comprensión, el uso y la gestión eficiente de estos elementos, así como para proporcionar información relevante a quienes trabajan con ellos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

La documentación de un sistema puede ser interna y externa: La primera se encuentra en las líneas de código dentro de nuestro programa, aquí se detalla que hace cada sección del código del programa, la segunda se encarga de un análisis, diagramas de flujo o seudocódigo, manuales de usuarios, manuales técnicos con instrucciones para ejecutar el programa e interpretar los resultados que obtenemos.

La documentación es muy importante para poder corregir errores futuros en nuestra aplicación o bien realizar actualizaciones del mismo, este último proceso se llama mantenimiento del programa y con cada actualización la documentación también se debe actualizar de esta manera se facilita ajustes posteriores.

Algo común es realizar el manejo de versiones[3] mediante 3 números: X.Y.Z y cada uno indica una cosa diferente

- El primero (X) se le conoce como versión mayor y nos indica la versión principal del software.

  Ejemplo: 1.0.0, 3.0.0.0

- El segundo (Y) se le conoce como versión menor y nos indica nuevas funcionalidades. Ejemplo 1.2.0, 3.3.0

- El tercero (Z) se le conoce como revisión y nos indica que se hizo una revisión del código por algún fallo. Ejemplo 1.2.2, 3.3.4.

Ahora que conocemos el significado de cada número, viene una pregunta importante: ¿cómo sabemos cuándo cambiarlos y cual cambiar?

- Versión mayor o X, cuando agreguemos nuevas funcionalidades importantes, puede ser como un nuevo módulo o característica clave para la funcionalidad.

- Versión menor o Y, cuando hacemos correcciones menores, cuando arreglamos un error y se agregan funcionalidades que no son cruciales para el proyecto.

- Revisión o Z, cada vez que entregamos el proyecto.

Dentro del control de versión por números podemos agregar una clasificación por estabilidad del proyecto.

Las opciones que tenemos para este control son: Alpha, Beta

- **Alpha** es una versión inestable que es muy probable que tenga muchas opciones que mejorar, pero queremos que sea probada para encontrar errores y poder

[3] La fuente cuenta con una extensa información sobre el control de versiones en el desarrollo de aplicaciones informáticas: ¿Cómo se deciden las versiones del software? | EDteam

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

poner a prueba funcionalidades en la mayoría de los casos podemos decir que está casi listo el producto. Ejemplo 1.0Alpha, 1.0.a.1, 1.0a2.

- **Beta** es una versión más estable que Alpha en la que contamos con el producto es su totalidad, y se desea realizar pruebas de rendimiento, usabilidad y funcionamiento en algunos módulos para ver cómo funciona bajo un ambiente no tan controlado. Aquí aparece el nombre de Beta Tester que escuchamos en el mundo del software. Ejemplo 2.0Beta, 2.0b,2.0b1.
- El siguiente paso es **RC (Release Candidate)**, que es el último toque fino del software antes de salir y después de pasar por Beta. Ejemplo: 3.0-RC o también 3.0-RC1.

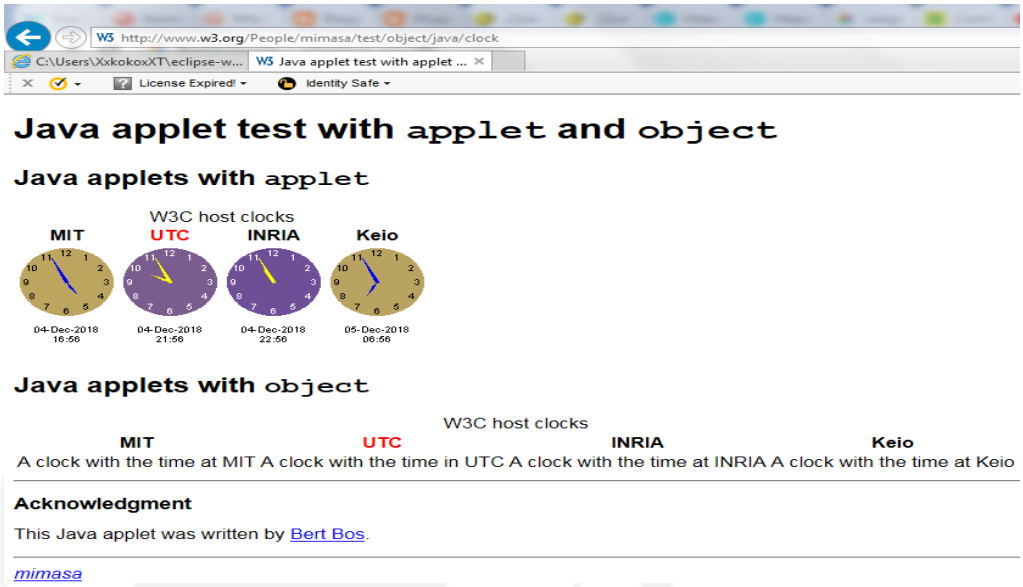
2.2. Creación de un Programa en Java

Java es un lenguaje de programación y plataforma informática con en cual podemos crear programas de cualquier tipo en sus diferentes plataformas. Esto hace de Java rápido, seguro, fiable y portable. En java podemos escribir básicamente 2 tipos de programas: applets y aplicaciones: y ambos requieren que nuestra computadora tenga instalado una versión del JMV (Java Virtual Machine) (Toro, 2018).

- **Applets.** Los applets son programas Java que se transmiten por internet y que se ejecutan incrustada dentro de una página web como se observa en la figura 10. Algo que destacar es que desde el 2014en adelante los diferentes navegadores como por ejemplo Google Chrome y Firefox han quitado el soporte al plugin de NPAPI

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 10
Ejemplo de un applet ejecutándose en un navegador web

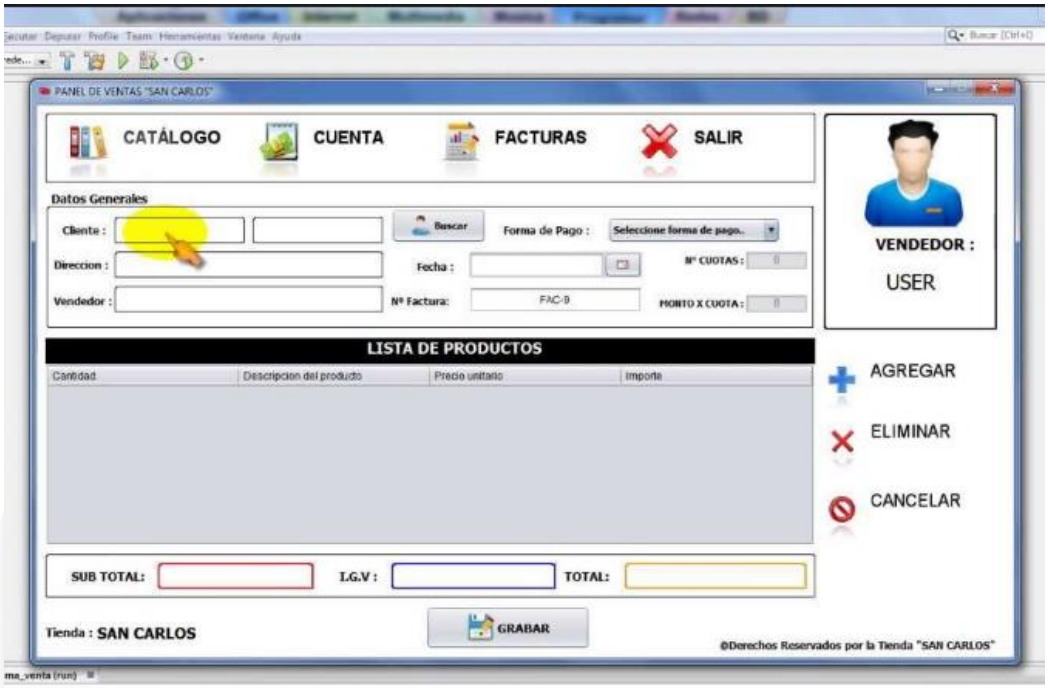


Nota. La figura representa como se muestra un applet desarrollado en Java. https://1.bp.blogspot.com/-BjGnHF2jYNU/Xat7jd5wWzI/AAAAAAAAF_w/wVu3AvDbc20xgNsZzQC1WUrkU8Q1eZLbwCKgBGAsYHg/s1600/image.png

- **Aplicaciones.** Las aplicaciones son programas Java que son independientes de un navegador web, poseen ventana propia a diferencia de los applets pero que también necesitan de una versión de la JMV para poder se ejecutados en el sistema operativo (Toro, 2018).

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 11
Aplicación Java de un sistema de ventas

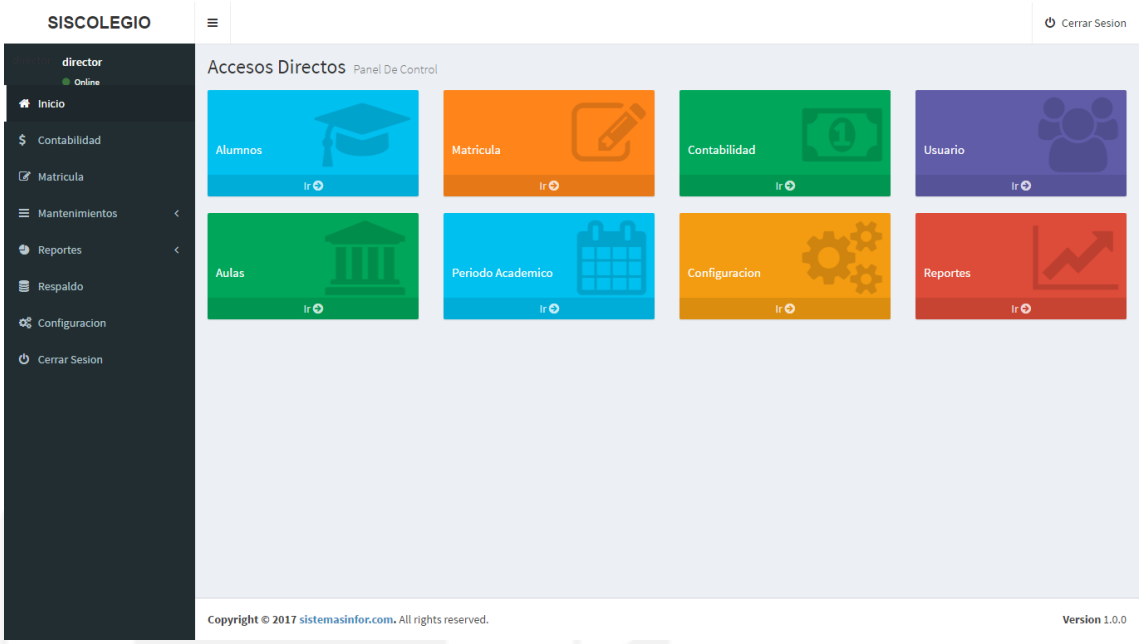


Nota. La figura representa como se muestra una aplicación de escritorio desarrollado en Java SE. https://2.bp.blogspot.com/-LyoSzLLLoaI/XauDScW05WI/AAAAAAAAGAY/sMLUjJUzlCkNWE6cfSQflyAEXW3oqdIQgCKgBGAsYHg/s1600/image.png

- **Aplicaciones Web.** Para desarrollar aplicaciones Web en Java se debe unir un conjunto de tecnologías como, por ejemplo. Servlets, JavaBeans, Java Server Page (JSP), Java Server Face (JSF) entre otras. Cuando se desarrolla aplicaciones Web en Java se trabaja con el paradigma **request-response** a diferencia de los applets las aplicaciones Webs se ejecutan del lado del servidor que interactúan con los clientes.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 12
Sistema Web de matrícula escolar



Nota. La figura representa como se muestra una aplicación web desarrollada en Java usando un distinto conjunto de tecnologías (imagen tomada del sistemainfor). https://2.bp.blogspot.com/-rn3ZXtX3SuE/WP5V84bGF7I/AAAAAAAAA8E/odZokM_eoFUTxGkNxHbWw2bm6lba5secACLcB/s1600/principal.png

- **Aplicaciones para dispositivos móviles, PDAs, electrodomésticos.** Para el desarrollo de aplicaciones también se debe unir distintas tecnologías estas aplicaciones están desarrolladas para ser ejecutadas en dispositivos móviles inteligentes, y electrodomésticos.

Como ya hemos detallado los dos tipos de programas que se pueden desarrollar en Java, en este libro nos vamos a centrar en el desarrollo de aplicaciones en lugar de applets, eso nos permitirá centrarnos y dirigir esfuerzos hacia los conceptos importantes de programación y en la resolución de problemas durante el proceso de aprendizaje.

Cuando se realiza un programa en Java este consta de una colección de clases, esta característica se cumple en los tipos de programas que se pueden desarrollar en Java; las clases contiene datos y métodos para su manipulación. En el siguiente ejemplo vamos a detallar un ejemplo de una aplicación simple en Java detallando como se ejecuta el mismo.

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola mundo");
    }
}
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Al momento que ejecutamos el programa y si el compilador no muestra ningún error, en consola se visualizara lo siguiente:

Figura 13
Salida en pantalla del programa en Java

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Co
Hola mundo

Process finished with exit code 0
```

Nota. La figura representa la ejecución de un programa en Java la salida se lo realiza en consola. Captura de pantalla.

La salida del programa es un Hola Mundo esto se debe a la instrucción println lo que hace esta instrucción es enviar el mensaje contenido entre comillas a la pantalla. El proceso de la ejecución del programa lo veremos en la ilustración de la figura 14.

1. **Edición del programa.** Para este paso se puede utilizar cualquier editor de texto como Word, Notepad entre otros dependiendo el Sistema Operativo que se utilice siempre y cuando sigamos las reglas de sintaxis de Java que se describirá en los siguientes capítulos. Todo el código de este programa se le llamara programa fuente. En el caso del ejemplo anterior se lo llamo **HolaMundo.java**

   Normalmente para la codificación de las aplicaciones en Java se utiliza entorno de desarrollo integrados que facilita la codificación y desarrollo del software. Estos entornos de desarrollo integrados cuentan con editores donde podemos editar el código, compiladores, depuradores para la detección de errores de sintaxis y un programa que carga el código para ser ejecutado.

2. **Compilación de un programa en Java.** Después de haber escrito el código fuente y antes de compilar el programa debemos revisar la sintaxis del código y verificar si cumple con todas las reglas de Java. Una vez verificado, se procederá a compilar nuestro programa con el compilador **javac**, en donde se encarga de buscar errores, si no encuentra ningún error traduce el programa a bytecode; el programa se guarda en un archivo con la extensión **.class.** Tomando el ejemplo anterior, el archivo fuente HolaMundo.java se convirtió y se almaceno como HolaMundo.class.

3. **Carga de un programa en la memoria.** Para ejecutar una aplicación en Java, lo que se hace es cargar el archivo en la memoria de la computadora. Cuando se escriben programas en Java como ya lo habíamos dicho se necita un entorno de desarrollo integrado llamados (IDE, por sus siglas en inglés), estos entornos

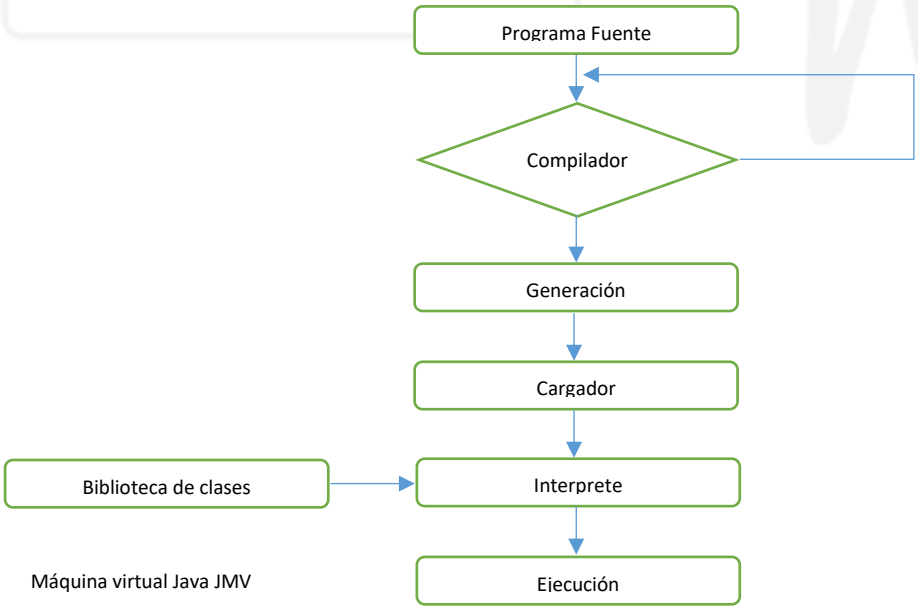
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

contienen bibliotecas de clases estas clases son programas que realizan tareas específicas, y también se cargan las bibliotecas desarrolladas por el usuario, que en Java se denomina paquetes(**packages**) que son un conjunto de clases que se relacionan entre sí. Para que la aplicación en Java se ejecute con éxito es necesario contar con otro programa llamado cargador (*loader*) este se encarga de enlazar y combinar todas las clases en bytecodes.

4. **Verificación de bytecodes y ejecución.** Ya que el programa cargador haya conectado los bytecode de las diferentes clases y el de su programa se carga en la memoria RAM. A medida que las clases se vayan cargando en la memoria principal un verificador de bytecodes aprueba que las clases están correctas y validas y no rompen ninguna restricción de seguridad de Java. Por último, el intérprete traduce cada una de las instrucciones de bytecode al lenguaje de máquina de su computadora independientemente del Sistema Operativo que se esté utilizando y enseguida se ejecuta.

Hoy en día las aplicaciones en Java se ejecutan más rápido ya que las máquinas virtuales Java han mejorado rotundamente y por lo general realizan simultáneamente combinaciones de interpretación, compilación inmediata, por tal motivo el proceso de ejecución se redujo considerablemente.

Figura 14
Diagrama de Flujo de un programa en Java



Nota: La figura de representa el diagrama de flujo de la ejecución de un programa en Java.

2.3. Metodología de la Programación

Cuando se desarrolla y se diseña software en Java, existen dos enfoques para diseñarlos: programación estructurada y programación orientada a objetos.

2.3.1. Programación Estructurada

El método de la programación estructurada utiliza métodos tradicionales de programación, estos métodos se van utilizando desde la década de los 60 y 70, especialmente desde la creación de lenguajes de programación como Pascal por Niklaus Wirth. Con este tipo de programación se tiene un enfoque especifico en el cual se puede crear programas muy bien escritos y legibles. Con este tipo de método se escriben programas de acuerdo con ciertas reglas y técnicas.

Las reglas de programación4 o diseño estructurado se basan en la modularización, a su vez cada módulo se analiza para obtener una solución individual, lo cual significa que la programación estructurada tiene un diseño descendente.

Las técnicas de programación estructurada incluyen construcciones, estructuras o instrucciones básicas de control estos conceptos lo revisaremos en capítulos siguientes, los cuales son:

- Secuencias
- Decisiones
- Bucles

La estructura básica de este método de programación indica el orden en el cual se van a ejecutar las instrucciones de un algoritmo o un programa.

Este método de programación se utiliza para programas pequeños, estos paradigmas o principios de organización resultan muy eficientes, lo que hace el programador es crear un conjunto de instrucciones en cierto lenguaje de programación, compilar y ejecutar ese conjunto de instrucciones, como ya lo habíamos dicho este método de programación solo funciona en programas pequeños, pero a medida que la dificultad de la solución del problema va siendo más complejo se va a utilizar otros tipos de métodos como la creación de métodos o funciones, en el cual se desglosa en unidades más pequeños.

[4] LA siguiente fuente con tiene un capítulo completo (1) donde se analiza y comparan con detalle ambos tipos de métodos de programación: Joyanes, L. y Sánchez, L. Programación en C++. Un enfoque práctico. Madrid MacGraw-Hill, 2006

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

2.3.2. Programación Orientada a Objetos

La programación orientada a objetos (POO)5 es un paradigma de la programación que se basa en el concepto de "objetos" para moldear el mundo real y resolver problemas de programación.

La Programación Orientada a Objetos (POO) es un enfoque de desarrollo de software que se basa en la creación y manipulación de objetos, que son instancias de clases. En POO, los objetos representan entidades del mundo real y tiene atributos(datos) y métodos(funciones) que operan en esos datos.

El objetivo principal de la POO es organizar el código de manera modular y reutilizable al permitir la encapsulación de datos y la abstracción de comportamientos. Esto facilita la creación de sistemas complejos al dividirlos en objetos interconectados que colaboran que colaboran para cumplir con sus responsabilidades y al mismo tiempo, proporciona un alto nivel de modularidad, flexibilidad y mantenibilidad en el desarrollo del software.

En Java, la creación de operaciones requiere la escritura de los algoritmos correspondientes y su implementación. En el enfoque orientado a objetos, las múltiples operaciones necesarias de un programa utilizan métodos para implementar los algoritmos, los cuales, a su vez utilizarán instrucciones o sentencias de control, de selección, repetitivas o iterativas (Joyanes Aguilar & Zahonero Martinez, 2011).

2.4. Metodología de Desarrollo Basada en Clases

En Java, una metodología de desarrollo basada en clases se refiere a la práctica de utilizar clases como el componente central y fundamental para organizar y estructura el código de un programa. La programación orientada a objetos (POO) es la base de esta metodología en Java. Detallaremos algunos elementos los cuales vamos a utilizar en esta metodología de programación.

- **Clases.** Las clases son plantillas o moldes que define la estructura y el comportamiento de los objetos.
- **Objetos.** Los objetos son instancias concretas de una clase. Se crean a partir de una clase y contiene daros específicos y pueden ejecutar métodos definidos en esa clase.
- **Encapsulamiento.** La encapsulación es el concepto de ocultar los detalles internos de una clase y proporcionar un interfaz publica para interactuar con los objetos.

[5] OOP. Object Oriented Programming

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **Herencia.** La herencia es un mecanismo que permite crear nuevas clases basadas en clases existentes.
- **Polimorfismo.** El polimorfismo permite que los objetos de diferentes clases se comprometen de manera similar al responder a un mismo conjunto de mensajes o métodos.
- **Abstracción.** La abstracción implica la creación de clases y objetos que representan conceptos abstractos y generalizados, ocultando los detalles innecesarios para centrarse em lo esencial.
- **Modularidad.** La modularidad se refiere a la división del programa en módulos o paquetes, donde cada módulo se relaciona con una clase o un conjunto de clases relacionadas.
- **Políticas de buenas prácticas.** Se siguen pautas y convenciones de codificación, como el uso de nombres de clases y métodos descriptivos, comentarios adecuados y diseño coherente, para asegurar que el código sea legible y mantenible.

Todos estos componentes lo vemos con más detalle en capítulos siguientes.

2.5. Entornos de Programación en Java

Aprender a programar en Java se necesita aprender y conocer ciertas técnicas y metodologías para realizar el análisis, diseño y construcción del software.
Para realizar el aprendizaje practico se realiza de diferentes maneras y con diferentes herramientas que hacen la programación más fácil como:

- JDK Java Development Kit
- IDE entono de desarrollo integrado
- Editores de texto, compiladores/depuradores.

Estas herramientas hacen más fácil al programador al realizar un programa en Java. Lo que el programador debe hacer es conocer a profundidad la herramienta que va a utilizar es por eso que en este capítulo terminaremos de ver que es o que necesitamos para poder programar en Java.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

2.5.1. El Kit de Desarrollo Java: Jdk 20

El JDK es una herramienta que sirve para crear programa en Java fue creada por Sun Microsystem, que consta de un conjunto de programas de líneas de ordenes que se utilizan para crear, compilar y ejecutar programas en Java, cada nueva versión de Java se acompaña por el kit de desarrollo, al momento que de escribir este capítulo, la última versión del JDK es la 20.

"Durante más de 20 años, Java[6] ha permitido a los desarrolladores diseñar y construir la próxima generación de aplicaciones robustas, escalables y seguras", afirmó Goeorges Saab, vicepresidente senior de desarrollo de la plataforma Java y presidente de la Junta Directiva de OpenJDK. "Las nuevas e innovadoras mejoras en Java 20 reflejan la visión y los invaluables esfuerzos que la comunidad global de Java a contribuido a lo largo de la existencia de Java".

El ultimo kit de desarrollo de Java (JDK) proporciona actualizaciones y mejoras con siete propuestas de mejora de JDK(JEP). La mayoría de las actualizaciones son funciones de seguimiento que mejoran la funcionalidad introducida en versiones anteriores.

JDK 20 ofrece mejoras de lenguaje del proyecto OpenJDK Amber (Record Patterns and Pattern Matching for Switvh); mejoras de OpenJDK Project Panama para nterconectar Java Virtual Machine (JVM) y código nativo (Foreign Function & Memory API y Vector API); y características relacionadas con Project Loom(Valores con alcance, subprocesos virtuales y concurrencia estructurada), que agilizarán drásticamente el proceso de escritura, mantenimiento y observación de aplicaciones concurrentes de alto rendimiento.

"Hoy en día, las organizaciones enfrentan una presión cada vez mayor para utilizar sus recursos de la manera más inteligente y eficiente posible, lo que requiere que los desarrolladores busquen herramientas que agilicen el desarrollo de aplicaciones y al mismo tiempo ayuden a garantizar que sus organizaciones alcancen sus objetivos de cumplimiento y seguridad de TI", dijo Jay Lyman, analista de investigación senior de S&P.

Las actualizaciones más importantes entregadas en Java 20 son:

Actualizaciones y mejoras del idioma

[6] La fuente tiene como referencia la página oficial de Oracle sobre la actualización de JDK 20 para el desarrollo de aplicaciones. https://www.oracle.com/news/announcement/oracle-releases-java-20-2023-03-21/

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **JEP 432: Patrones de registro (segunda vista previa):** mejora el lenguaje Java al permitir a los usuarios anidar patrones de registro y tios de patrones para crear una forma poderosa, declarativa y componible de navegación y procesamiento de datos. Esto ayuda a aumentar la productividad de los desarrolladores al permitirles ampliar la coincidencia de patrones para permitir consultas de datos más sofisticados y componible.
- **JEP 443: Coincidencia para patrones para Switch(cuarta vista previa):** al extender la coincidencia de patrones para switch, se puede probar una expresión con varios patrones, cada uno con una acción especifica, de modo que las consultas complejas orientadas a datos se puedan expresar de manera concisa y segura. Ampliar la expresividad y aplicabilidad de las expresiones y declaraciones de cambio ayuda a aumentar la productividad de los desarrolladores.

Características de la incubadora/vista previa del telar del proyecto

- **JEP 429: Valores de ámbito (incubadora):** permite compartir datos inmutables dentro y entre subprocesos, que se prefieren a las variables locales de subprocesos, especialmente cuando se utilizan una gran cantidad de subprocesos virtuales. Esto aumenta la facilidad de uso, la comprensibilidad, la solidez y el rendimiento.
- **JEP 436: Subprocesos virtuales (segunda vista previa):** agilice significativamente el proceso de escritura, mantenimiento y observación de aplicaciones concurrentes de alto rendimiento mediante la introducción de subprocesos virtuales livianos en la plataforma Java. Al permitir a los desarrolladores solucionar problemas, depurar y perfilar fácilmente aplicaciones simultáneas con herramientas y técnicas JDK existentes, los subprocesos virtuales ayudan a acelerar el desarrollo de aplicaciones.
- **JEP 437: Concurrencia estructurada (segunda incubadora):** simplifica la programación multiprocesos al tratar multitareas que se ejecutan en diferentes subprocesos como una sola unidad de trabajo. Esto ayuda a los equipos de desarrollo a optimizar el manejo y la cancelación de errores, mejorar la confiablidad y mejorar la observabilidad.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Características de la vista previa del proyecto Panamá.

- **JEP 434: API de memoria y funciones externas (segunda vista previa):** permite a los programas Java interoperen con código y datos fuera del tiempo de ejecución de Java. Al invocar eficientemente funciones externas (es decir, código fuera de la máquina virtual Java [JVM]) y al acceder de forma segura a memoria externa (es decir memoria no administrada por la JVM), esta característica permite a los programadores Java llamar a bibliotecas nativas y procesar datos nativos sin que requiere la interfaz nativa de Java. Esto aumenta
- la facilidad de uso, el rendimiento y la seguridad.
- **JEP 438: API vectorial (Quinta incubadora):** expresa cálculos vectoriales que se compilan de manera confiable en tiempo de ejecución en instrucciones vectoriales en arquitecturas de CPU compatibles. Esto aumenta el rendimiento en comparación con los cálculos escalares equivalente.

El lanzamiento de Java 20 es el resultado de una amplia colaboración entre los ingenieros de Oracle y otros miembros de la comunidad mundial de desarrolladores de Java a través de OpenJDK y Java Community Process (JCP). Además de las nuevas mejoras, Java 20 cuenta con el respaldo de Java Management Service , un servicio nativo de Oracle Cloud Infrastructure (OCI), que proporciona un panel único para ayudar a las organizaciones a administrar tiempos de ejecución y aplicaciones de Java en las instalaciones o en cualquier nube.2.6. ENTORNOS DE DESARROLLO INTEGRADO (IDE)

2.6. Herramientas Para Desarrollo en Java

Antes de empezar a practicar y desarrollar aplicaciones en Java en su computadora, debemos disponer de todas las herramientas adecuadas, para poder editar, compilar y ejecutar los programas en Java, muchas de las herramientas que vamos a utilizar son gratuitas y las puede descargar desde el sitio oficial de los desarrolladores.

Existen distintas versiones y populares de entornos de desarrollo para Java como Apache Netbeans, Eclipse JBuilder, IntellijIDEA entre otros para el propósito de nuestro libro vamos a utilizar IntelliJIDEA.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

2.6.1. Apache Netbeans

Apache NetBeans, anteriormente conocido simplemente como “NeatBeans”, es un entorno de desarrollo integrado IDE de código abierto que se utiliza principalmente para desarrollar aplicaciones en Java, pero también admite varios otros lenguajes de programación, incluyendo PHP, HTML, JavaScript, c/C++, y más. Fue originalmente desarrollado por Sun Mycrosystems y más tarde adquirido por Oracle Corporation cuando esta última compró a Sun Mycrosystems. Sin embargo, en el 2016, Orcle donó el proyecto a la Apache Software Foundation, y desde entonces se conoce como Apache NetBeans.

Algunas de las principales características de Apache NetBeans incluyen:

1. Editor de código.
2. Gestión de Proyectos.
3. Depuración.
4. Soporte de Lenguaje múltiples.
5. Creación de interfaces gráficas.
6. Administración de dependencias.
7. Extensiones y complementos.
8. Compatibilidad y multiplataforma.

2.6.2. Eclipse

Eclipse es un entorno de desarrollo integrado IDE de código abierto ampliamente usado en la programación de software en Java. Fue inicialmente desarrollado por IBM en 2001 y luego donado a la Eclipse Foundation, una organización sin fines de lucro que se encarga de su desarrollo y mantenimiento. Eclipse es conocido por su flexibilidad y extensibilidad, lo que hace adecuado para una amplia variedad de lenguajes de programación y tipos de proyectos. Algunas de sus características son:

1. Editor de código.
2. Gestión de proyectos.
3. Depuración
4. Control de versiones.
5. Extensibilidad.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

6. Soporte para múltiples lenguajes.
7. Desarrollo de aplicaciones empresariales
8. Comunidad activa

2.6.3. Intellij Idea

Intellij IDEA es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains, una empresa de software con sede en Republica Checa. IntelliJ IDEA es uno de los IDE más populares y ampliamente utilizados en la industria del desarrollo de software, y se utiliza principalmente para la programación en lenguajes como Java, Kotlin, Groovy, Scala y otros.

Algunas de las características y ventajas clase de IntelliJ IDEA incluyen:

1. Potente editor de código.
2. Análisis estático.
3. Soporte para múltiples lenguajes.
4. Depuración avanzada
5. Gestión de proyectos
6. Pruebas integradas
7. Integración con control de versiones
8. Extensibilidad
9. Herramientas específicas de desarrollo
10. Interfaz de usuario amigable

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen del Capítulo 2

Este capítulo se centra en proporcionar una guía para el desarrollo de programas en el lenguaje de programación Java, haciendo énfasis en alas mejores prácticas y una metodología efectiva.

- **Metodología de Desarrollo:** Se presentan las etapas típicas de desarrollo de software, como el análisis, diseño, implementación, pruebas y mantenimiento del software. Se enfatiza la importancia de seguir un proceso estructurado.
- **Programación estructurada:** La programación estructurada es un paradigma de programación que se basa en la organización lógica y ordenada del código fuente.
- **Programación Orientada a Objetos:** La programación Orientada a Objetos POO es un paradigma de programación que se basa en el concepto de la creación de objetos y la interacción entre ellos.
- **Kit de desarrollo JDK 20:** JDK suele introducir mejoras en el lenguaje de Java. Estas mejoras pueden incluir nuevas palabras claves, funcionalidades de lenguaje, mejoras en la sintaxis, etc.
- **Entorno de desarrollo:** Se describe que tipos de entrono de desarrollo utilizar para la programación en Java, dando una clara perspectiva que tipo de IDE utilizar para el desarrollo de software

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Capítulo 3

Elementos Básicos en Java

3.1 Estructura General de un Programa en Java

En este capítulo vamos hablar sobre la estructura y los elementos de un programa en Java, describiendo ideas relativas a su estructura; cada programa en Java se compone de una o más clases obligatoriamente un programa debe tener una clase **main**, esta clase debe tener un método main(). Por otro lado, un programa en Java debe tener un conjunto de declaraciones **import** esto me permite importar bibliotecas o archivos de Java. De modo que un programa en Java debe incluir lo siguiente.

- Declaraciones para importar paquetes
- Declaraciones de clases
- El método main()
- Métodos definidos por el programador dentro de la clase
- Comentarios del programa (se utilizan en todo el programa)

La estructura completa de un programa en Java se muestra en la en la figura 15 que a continuación se detalla con un ejemplo sencillo de Java.

Figura 15
Código sencillo en Java

```
package hola;                                   ← Nombre del paquete donde está la clase
import java.io.*;                               ← Archivo de clases de entrada /salida

public class HolaMundo {                        ← Nombre de la clase principal
    public static void main(String[] args) {    ← Cabecera del método
                                                 ← Nombre del método

        System.out.println("HOLA MUNDO DESDE JAVA");   ← Sentencias

    }
}
```

Nota: La figura representa el código de un programa sencillo en Java explicando la estructura y a que representa cada una de las líneas de código. Captura de pantalla.

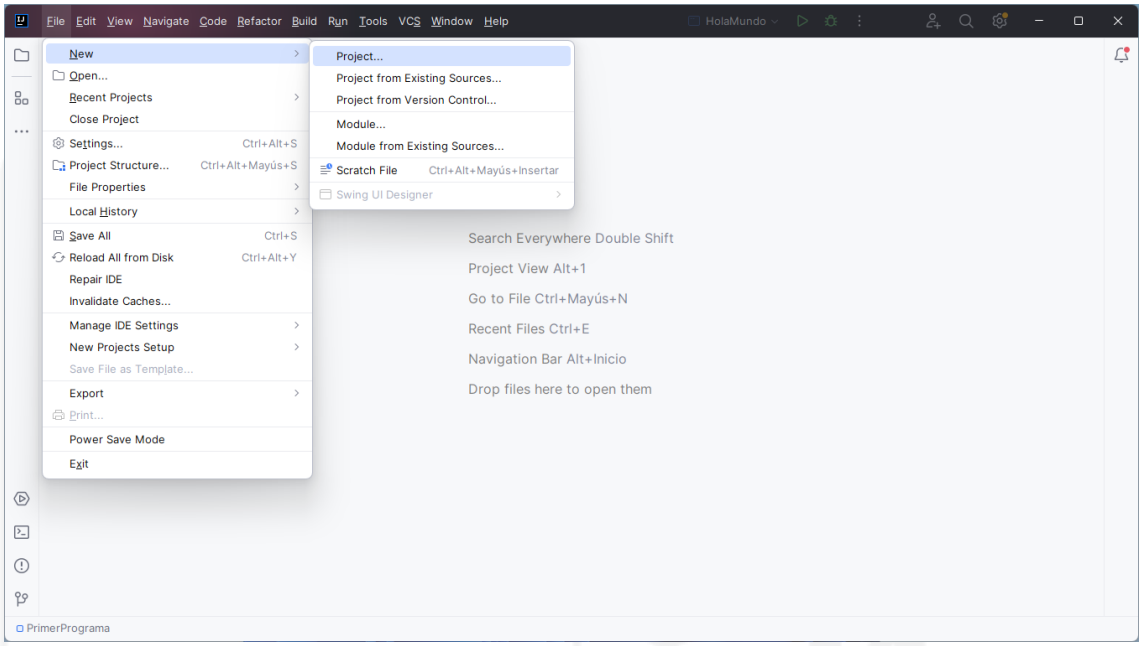
(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.1.1. Creando Mi Primer Programa en Intellij Idea

Para la parte práctica del libro utilizaremos el entorno de desarrollo integrado INTELLIJ IDEA la versión Community Edition que se lo descarga de la página oficial, https://www.jetbrains.com/es-es/idea/download/?section=windows.

1. Abrimos el software instalado, presionamos en File, new Project

Figura 16
Creando programa en Intellij IDEA

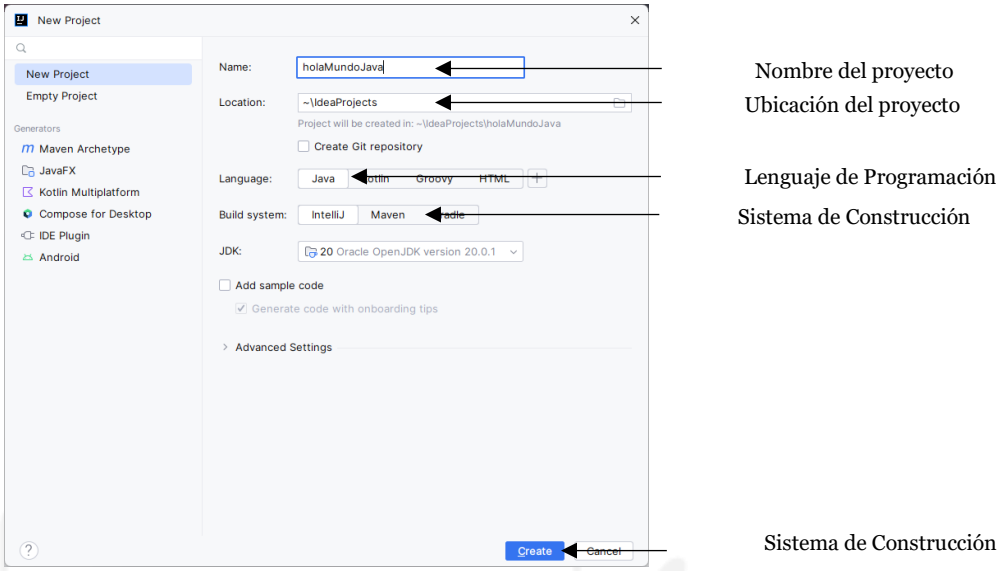


Nota: La figura representa como se crea un proyecto en Java con IntelliJ IDEA. Captura de pantalla.

2. Colocamos el nombre del proyecto y la ubicación donde se va a guardar el proyecto para el ejemplo holaMundoJava.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

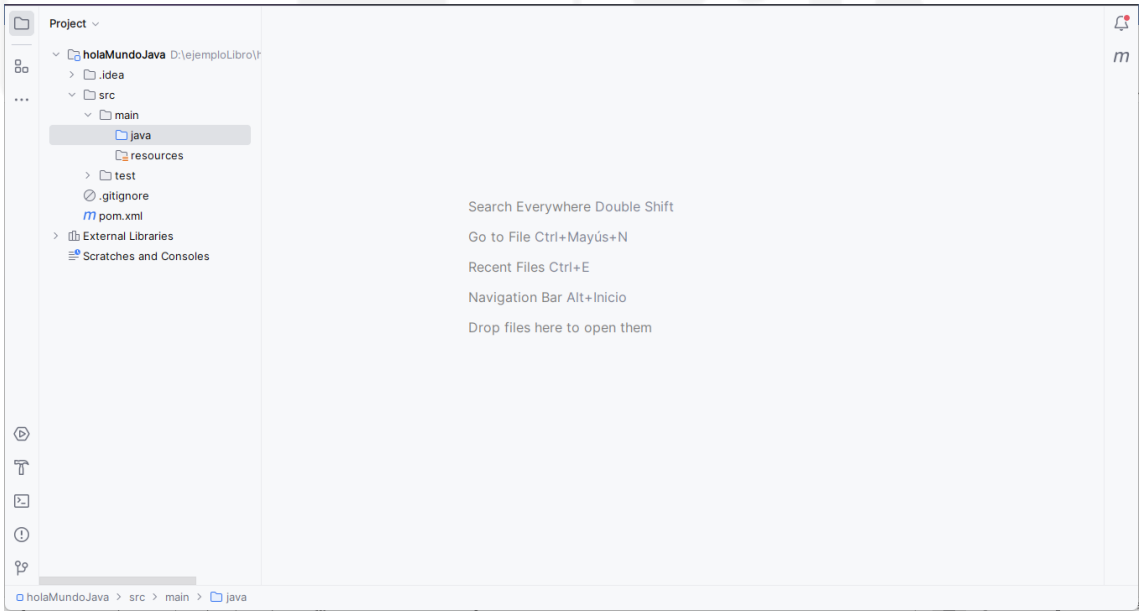
Figura 17
Creando Primer Programa



Nota: La Figura representa como crear mi primer proyecto en IntelliJ IDEA con lenguaje de programación en Java. Captura de pantalla.

3. Al Crear el proyecto, nos va a crear la estructura de nuestra aplicación de la siguiente manera.

Figura 18
Estructura de un Programa en Java

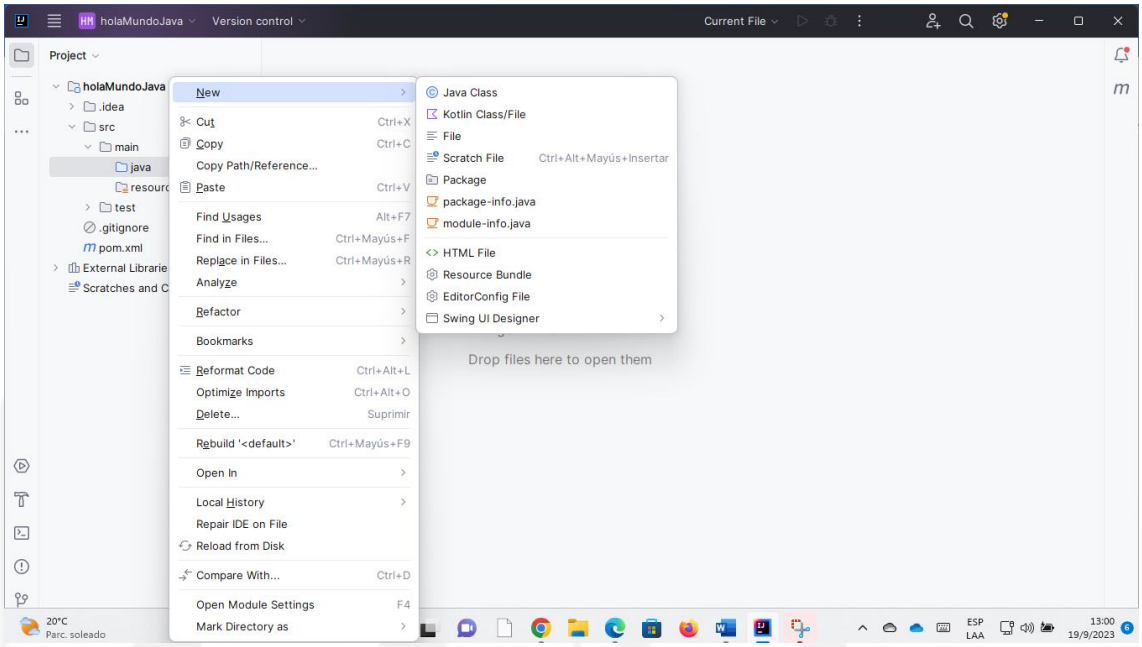


Nota: La figura corresponde a la estructura de nuestro programa. Captura de pantalla.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

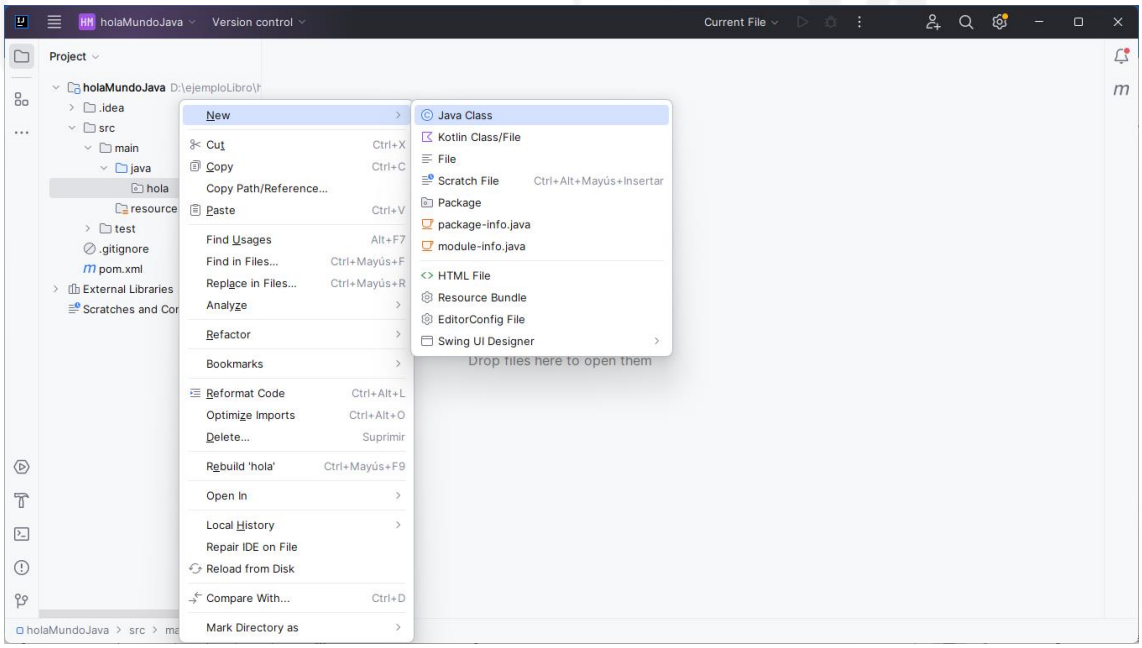
Dentro de la carpeta src, se encuentra la carpeta java ahí vamos a crear un paquete y una clase main().

Figura 19
Creación de un paquete



Nota: La figura representa la creación de un paquete. Captura de pantalla.

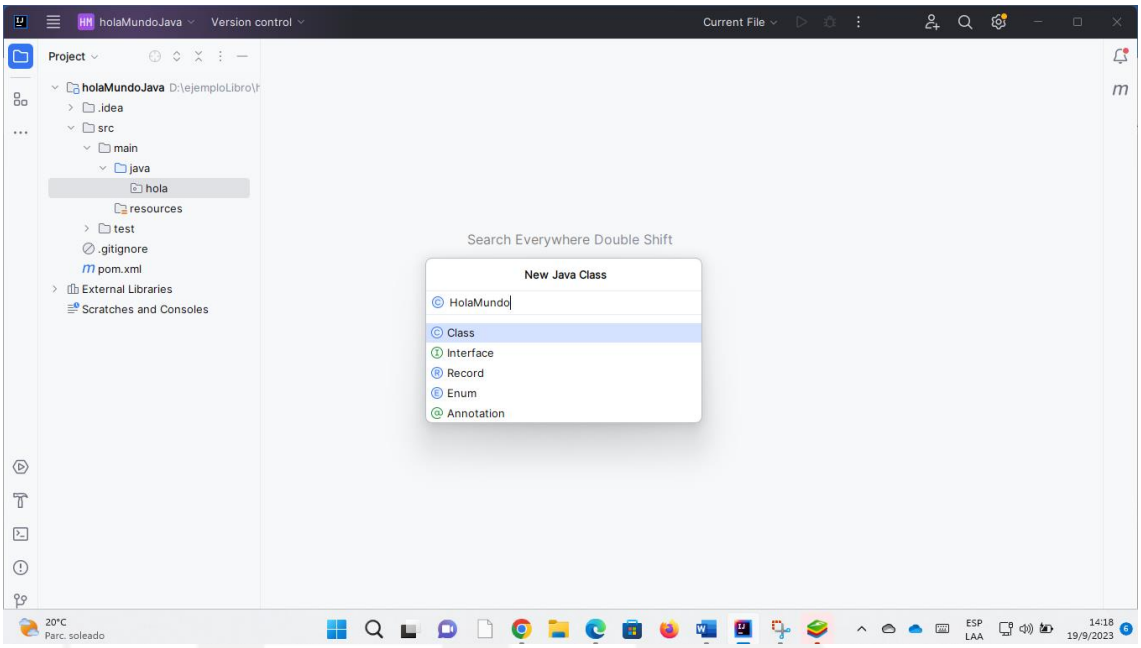
Figura 20
Creación clase Main



Nota: La figura representa como crear una clase. Captura de pantalla.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 21
Clase Hola Mundo



Nota: La presente figura se crea una clase de nombre Hola Mundo. Captura de pantalla

Una de las características al crear una clase en Java, esta se la debe nombrar con una letra mayúscula como se mira en la figura 19. No es una regla de Java, pero para los programadores es muy importante que las clases de nombren con una letra mayúscula al principio si el nombre de la clase contiene dos palabras cada palabra empieza con una letra mayúscula, tampoco debe llevar signos de puntuación.

La declaración del import que tenemos en la primera línea me permite importar librerías o paquetes de Java en este caso java.io proporciona entrada y salida del sistema a través de flujos de datos, serializados y el sistema de archivos[7] como se puede observar el (*) en la importación eso quiere decir que importamos todos los elementos que contiene el paquete java.io.

Cuando hablamos de comentarios en los lenguajes de programación, nos referimos a líneas en nuestro código que no se van a leer y no van a tener ningún cambio nuestro programa, estas líneas sirven para explicar lo que hace esa sección de código como podemos ver en la figura

[7] La siguiente fuente tiene como referencia sobre paquetes en Java
https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html

Figura 22
Comentarios en Java

```java
package hola;
import java.io.*;
/*Cuando escribimos un comentario en Java utilizamos
* dos // para comentar una sola linea
* pero para comentar una seccion utilizamos los signos  que
* pueden observar*/

public class HolaMundo {
    //aqui tenemos nuestro método main que va a permitir ejecutar mi programa
    public static void main(String[] args) {
        System.out.println("HOLA MUNDO DESDE JAVA");

    }
}
```

Nota: Como podemos ver en la figura en Java se puede representar los comentarios de dos formas una sección de código o comentarios de una sola línea. Captura de pantalla.

Un programa en Java se puede considerar un conjunto de clases, en la que al menos en una de esas clases debe tener un método main(), este método me permite ejecutar mi programa sin este método no tendríamos nada en salida de nuestra aplicación.

Figura 23
Método main()

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


}
```

Nota: La figura me muestra el código del método main() y dentro de las llaves va todo lo que se ejecuta

Import

En Java, todas las clases se agrupan en paquetes o packages que definen utilidades o grupos temáticos y que se encuentran en directorios del disco con su mismo nombre. Para incorporar y utilizar las clases de un paquete en un programa se utiliza la declaración ***import***: por ejemplo, si le queremos indicar al compilador que queremos importar la clase Math del paquete lang se debe escribir:

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura **24**
Sentencia import

```
import java.lang.Math;
```

Nota: La figura representa como se debe importar una clase de Java de un paquete que contiene dicha clase. Captura de pantalla.

La sintaxis general de la declaración import es:

```
import nombrePaquete.nombreClase;
```

nombreClase es el identificador de una clase del paquete; en un programa se puede incorporar varias clases con una secuencia de sentencias import; por ejemplo, para importar la clase BufferedREader, IOException, InputStreamReader del paquete java.io se debe escribir:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

Se puede especificar que se incorporen todas las clases publicas de un paquete, en este caso el nombreClase se sustituye por el *, de esta manera se puede utilizar todas las clases públicas del paquete en el programa.

```
import java.io.*;
```

EL programador puede definir sus propios paquetes y dentro podemos agrupar las clases que el programador desarrolle, de igual manera si en alguna parte del programa se necesita de esa clase simplemente se utiliza la sentencia import nombrePaquete.nombreClase. Ejemplo si en un proyecto tenemos un paquete de nombre auto y dentro del paquete se tiene un conjunto de clases como Automático, Manual, de esta manera hacemos las declaraciones

```
import auto.Automatico;
import auto.Manual;
```

Si deseamos llamar a todas las clases del paquete auto lo instanciamos de la siguiente manera:

```
import auto.*;
```

Las declaraciones de las importaciones de clases se lo realizan al inicio de la programación en la cabecera de nuestro programa y de esa manera se pueden utilizar esas clases en todo el programa sin necesidad de volverlas a llamar.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.1.2. Declaración de Clases

Como ya se ha dicho un programa en Java debe tener por lo menos una clase, la clase principal donde se incluya el método main(), y otros método y variables que el programador vea necesario; para declarar una clase debe llevar el nombre con una palabra clave el nombre de la clase debe empezar con una letra mayúscula y si el nombre de la clase lleva dos palabras deben estar unidas y cada palabra debe empezar con una letra mayúscula, generalmente indicando el acceso, seguida por su indicador, la palabra reservada class, su nombre y sus miembro por ejemplo:

```
package suma;

public class Suma {
    public static void main(String[] args) {
        int inicio, fin, suma;
        suma=0;
        inicio= 1;
        fin=100;
        for (int i = inicio; i < fin; i++) {
            System.out.print(i + " + " + suma + " = " );
             suma = suma + i;
            System.out.println(suma);
        }
    }
}
```

El archivo donde se va a guardar el programa del ejemplo anterior debe tener como nombre Suma.java, el nombre del archivo fuente siempre debe ser el mismo de la clase principal, es decir, la que contiene main(), y la extensión java.

3.1.3. Método (main)

El método main() es el punto de entrada esencial para la ejecución de programas en Java. En esencia, es donde inicia la acción de un programa en Java. Como habíamos hablado anteriormente en un programa desarrollado en Java, por lo menos debe haber una clase que tenga una método main().

```
public static void main(String[] args) {
    ...  bloque de sentencias
}
```

El método main() se declara como público *public,* lo que significa que es accesible desde cualquier parte del programa, se lo debe declarar como estático *static,* lo que implica que no necesitas crear una instancia de la clase que contiene el método *main* para utilizarlo. Puedes invocarlo directamente en la clase.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

El método *main()* tiene un tipo de retorno *void*, lo que indica que no devuelve ningun valor. En lugar de retornar algo, su propósito es iniciar la ejecución del programa, el método toma un parámetro especial llamado *args,* que es un arreglo (array) de cadena de texto. Este parámetro permite que el programa reciba argumentos desde la línea de comandos, lo que facilita la interacción con el usuario o la transmisión de información adicional al programa.

El cuerpo del método *main se* define entre llaves *{}* y es donde se coloca las instrucciones que componen el funcionamiento del programa

3.1.4. Métodos Definidos por el Usuario

Los métodos definidos por el usuario en Java son funciones personalizadas creadas por el programador dentro de un programa Java para realizar tareas específicas. Estos métodos son una manera de organizar y reutilizar el código al encapsular un conjunto de instrucciones bajo un nombre único. Pueden aceptar parámetros como entrada de datos y devolver resultados, lo que permite una modularidad y claridad efectivas en la programación. Estos métodos son esenciales para dividir un programa en partes más pequeñas y manejables, lo que facilita su comprensión y mantenimiento.

Los métodos definidos por el usuario se invocan, dentro de la clase donde se definieron, por su nombre y los parámetros opcionales que pudieran tener, después de que el método se invoca, el código asociado se ejecuta, y a continuación se retorna el método llamador. Si la llamada es desde un objeto de la clase, se invoca el método precedido del objeto y el selector punto (.). En capítulos más adelante se verá a fondo como instanciar y llamar a un método de una clase en Java, mientras tanto a título de ejemplo tomando el ejemplo anterior vamos a crear un método de nombre suma y lo llamamos en nuestro método *main()*.

```
package suma;
public class Suma {
    public static void main(String[] args) {
        //Instanciamos el objeto Suma
        Suma op= new Suma();
        /*
        * LLamamos al método suma para poder
        * ejecutar la operación de sumar los números
        * del 1 al 100
        * */
        op.operacion();
    }

    //método donde estamos realizando la operación
    public void operacion(){
        int inicio, fin, suma;
        suma=0;
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
        inicio= 1;
        fin=100;
        for (int i = inicio; i < fin; i++) {
            System.out.print(i + " + " + suma + " = " );
            suma = suma + i;
            System.out.println(suma);
        }
    }
}
```

Como se ve en el ejemplo el programador ha creado un método llamado operación(), este método no devuelve nada ya que utiliza la palabra reservada *void* dentro del método declaramos tres variables de tipo entero, inicio, fin y suma; cada una de estas están inicializada después generamos un bucle *for* que inicia en 1 y termina en 100 y se va recorriendo 1 a 1, dentro del for vamos a sumar cada uno de los números del 1 al 100.

Los métodos en Java se especifican en la clase a la cual pertenecen, la estructura de un método es la siguiente:

```
tipo_retorno nombreMetodo(lista_de_parametros){  principio del método
     sentencias                    cuerpo del método
    return expresión              valor que devuelve el método
}                                 fin del método
tipo_retorno                      Es el tipo de valor o void devuelto por la función
nombreMetodo                      Nombre del método
lista_de_parametros              Lista de parámetros, o void pasados    al método
                                  se los conoce también como argumentos
```

Ejemplo 3.1

Realizar un programa en Java formado por una clase que contenga dos métodos además del *main(): calcula(), mostrar(),* el primer método determina el valor de una función matemática para un valor dado de x, el segundo método muestra el resultado, *el método main()* llama a cada uno de los métodos auxiliares, los métodos deben ser estáticos por las restricciones de *main()*, la función evalúa utilizando el coseno de la clase Math, que esta se encuentra en el paquete java.lang.

```
package evaluar;

public class Evaluar {
    public static void main(String[] args) {
        double f;
        f=calcular();
        mostrar(f);

    }
    public static double calcular(){
        double x=3.141516/4;
        return x*Math.cos(x) + 0.5;
    }
    public static void  mostrar(double r){
        System.out.println(" Valor de la función r = " + r);
    }

}
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.1.5. Comentarios

Como ya hemos explicado anteriormente, los comentarios es información que no va a leer el programa, los comentarios son líneas en nuestro código donde explicamos que hace la sección de código, estas líneas son ignorados por le compilador y no realiza ningún proceso.

En programación la implementación de comentarios es una buena practica al momento de programar, como desarrolladores debemos comentar el código tanto como sea posible, de esta manera usted mismo y otros programadores pueden leer fácilmente lo que se esta desarrollando. En Java se suele comentar los programas al inicio de cada archivo fuente, estos comentarios al inicio se detalla que hace el programa, quien es la persona que la desarrollo y la fecha que fue creado el programa e información de revisión.

En Java los comentarios de un programa se pueden introducir de dos formas:

- Con los siguientes caracteres */*....*cometarios*....**/* para insertar más de una línea
- Con la secuencia de dos backslash o barras (//) me sirve sola para incorporar el comentario en una sola línea.

Comentarios con /* */

Los comentarios comienzan y terminan con la secuencia /* */; el texto va entre los caracteres y todo lo que esta dentro es ignorado por el compilador

/*Saludo.java Primer programa en java*/

Si se necita introducir varias líneas de comentarios de programas se puede hacer de la siguiente manera.

```
/*
* Programa: Evaluar.java
* Programador: Elvis Pachacama
* Descripción: Primer programa en Java
* Fecha de creación: 25 septiembre del 2023
* Revisión: Ninguna
* */
```

También los comentarios lo podemos introducir de la siguiente manera.

```
System.out.println(" Valor de la función r = " + r);/*Imprime el valor
de la función*/
```

Cometario en una línea con //

Con este tipo de comentarios de igual manera el compilador no lo va a ejecutar y se inserta en una sola línea

```
//Evaluar.java
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Ejemplo 3.2

Crear un programa en Java donde se guarde un mensaje en una variable de tipo String y lo imprima en la pantalla, el programa

```
package mensaje;
/*
* Programa: Mensaje.java
* Programador: Ing. Elvis Pachacama
* Descripción: Escribir un programa donde se guarde una mensaje en una
variable de tipo String
*              imprimir el valor de la variable en pantalla.
* Fecha de creación: 25 Septiembre del 2023
* Revisión: ninguna
* */
public class Mensaje {
    public static void main(String[] args) {
        String mensaje = "Escribiendo un programa en Java...";
        System.out.println(mensaje);
    }
}
```

3.2. Elementos de un Programa en Java

Todos los programas escritos en Java constan de un archivo donde se encuentran las clases y métodos que escribe el programador, además de otros archivos que se encuentran incorporadas en otras clases, el compilador traduce todos programas y además incorporan todas las clases solicitadas al programa y se encarga de analizar la secuencia de los tokens.

3.2.1. Elementos Lexicos del Programa

Cunado se programa en Java encontraremos 5 clases de tokens como son: identificadores, palabras reservadas, literales, operadores y otros separadores, los cuales vamos air identificando con el pasar del libro.

Identificadores

Los identificadores es un conjunto de caracteres que se utilizan para representar las variables, métodos, clases y otros elementos del código fuente. Los identificadores siguen ciertas reglas y convenciones que deben cumplirse.

- **Caracteres permitidos.** Los identificadores pueden contener letras mayúsculas, minúsculas, dígitos y el carácter especial "_" (guion bajo).
- **No pueden empezar con un número.** Un identificador no puede empezar con un número, pero si puede contener un número después del primer carácter.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **Sensible a mayúsculas y minúsculas.** Java distingue entre mayúsculas y minúsculas en los identificadores, ejemplo **"miVariable"** y **"MiVariable",** como se ve en el ejemplo el nombre es el mismo, pero en Java son distintos identificadores.
- **No se pueden usar palabras claves.** No puedes utilizar palabras claves reservadas de Java como identificadores. Ejemplo **"public"**, **"class", "int", "if",** entre otras clases.

Cuando se usa identificadores estos pueden ser creados de cualquier longitud, Java no te limita en el número de caracteres que puede contener. Consejos que te pueden servir de reglas para escribir un identificador.

1. Identificadores de variables con minúsculas.
2. Constantes con mayúsculas.
3. Métodos en minúsculas.
4. Clases con el primer carácter en mayúsculas.

Palabras Reservadas

Cuando se programa en Java hay palabras reservadas, las cuales no puedes ser usadas para nombrar una clase, método o variable, esta es una característica de Java ya que estas palabras se asocian con algún significado especial ejemplo.

```
//error no se puede nombrear un método con una palabra reservada
void void(){    //error
    //error no se puede nombrar una variable con una palabra reservada
    int char;  //error
}
```

En Java actualmente existen 53 palabras[8] reservadas.

Tabla 1
Palabras reservadas en Java

abstract	**continue**	**for**	**new**	**switch**
assert	**default**	**goto**	**package**	**synchronized**
boolean	**do**	**if**	**private**	**this**
break	**double**	**implements**	**protected**	**throw**

[8] La siguiente fuente tiene como referencia las palabras reservadas en Java las que no se pueden usar en un programa para nombrar un método, clase, etc.
https://codigofacilito.com/articulos/palabras_reservadas_java

byte	**else**	**import**	**public**	**thows**
case	**enum**	**instanceof**	**return**	**transient**
catch	**extends**	**int**	**short**	**try**
char	**final**	**interface**	**static**	**void**
class	**finally**	**long**	**strictfp**	**volatile**
const	**float**	**native**	**super**	**while**
true	**false**	**null**		

Dos de las palabras que tenemos en la tabla anterior hoy en día ya no son utilizadas en Java; *const y goto,* estas palabras son reservadas del lenguaje de programación C++, estas palabras se mantuvieron para generar mejores mensajes de error.

3.2.2. Paquetes

Como ya lo habíamos dicho Java agrupa las clases e interfaces que tienen cierta relación mediante archivos especiales las cuales contienes declaraciones de clases con sus métodos, estas agrupaciones lógicas de clases y otros tipos relacionados que nos permiten organizar y estructurar un programa en Java de manera más eficiente. Estos paquetes ayudan a evitar conflictos de nombres y hacen que el código se más modular y fácil de mantener. Cada paquete en Java actúa como un contenedor que contiene clases y otros recursos relacionados. Los paquetes se organizan en una jerarquía, y el nombre completo de una clase incluye la ruta del paquete en la que se encuentra.

Java también proporciona paquetes ya desarrollados, y los más importantes son:

Java.lang, java.applet, java.awt, java.io y java.util.

java.lang, es un conjunto de clases y recursos esenciales que proporcionan funcionalidades, básicas y fundamentales para la programación en Java. Estas clases incluyen objetos fundamentales como ***String, Integer, Boolean, Math.***

Java.io, es un conjunto de clases y utilidades que proporciona funcionalidades para la entrada y salida de datos (E/S). Ofrece clases como: ***FileInputStream, FileOutputStream, BufferesReader, BufferedWriter.***

Java.util, es un conjunto amplio y diverso de clases y utilidades que proporciona funcionalidades adicionales y utilidades de propósito general para la programación en Java. Contiene clases para estructuras de datos como listas, conjuntos, mapas, así como

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

clases para manejar fechas, tiempos, y muchas otras utilidades útiles para tareas comunes de programación.

Java.awt, que significa "Abstract Window Toolkit" en inglés, es un conjunto de clases y utilidades en Java diseñado para facilitar la creación de interfaces gráficas de usuario (GUI). Este paquete proporciona una variedad de componentes figuras, como botones, ventanas, cuadros de texto y otros elementos, que permiten a los desarrolladores construir aplicaciones Java con interfaces de usuario interactivas y visualmente atractivas.

En esencia, java.awt es una parte esencial de la biblioteca estándar de Java que facilita la creación de aplicaciones con GUI, ayudando a los programadores a diseñar interfaces de usuario intuitivas y portables que pueden ejecutarse en diferentes sistemas operativos sin cambios significativos en el código fuente.

No olvidemos que el programador puede crear sus propios paquetes y después puede utilizarlo en cualquier parte de la aplicación que desee. Ejemplo tenemos una clase *Auto* y se va almacenar en el paquete *vehículo*.

```
package vehiculo;

public class Auto {
    double precio;
    public static void conducirVehiculo(){

    }

}
```

Ya creado el directorio, Java generar un subdirectorio con el mismo nombre del paquete *vehiculo,* dentro de este paquete tenemos la clase *Auto,* el programador puede utilizar la clase creada del paquete *vehiculo,* anteponiendo en el nombre del paquete al nombre de la clase. El compilador lo que hace es buscar el archivo de la clase en el paquete: *vehiculo.Auto.conducirVehiculo();*

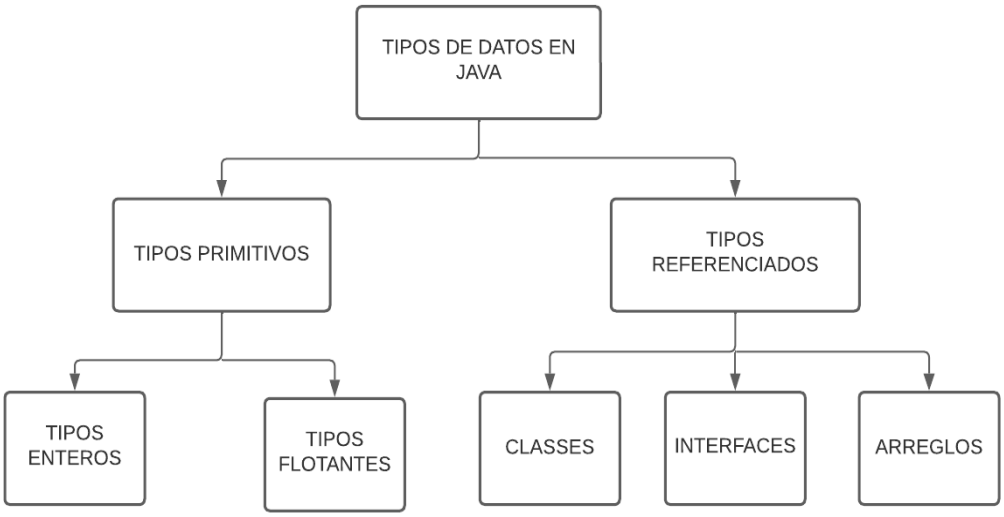
3.3. Tipos de Datos en Java

Los tipos de datos en Java se refieren a las categorías de valores de valores que las variables pueden contener. Estos tipos definen qué tipo de información puede almacenarse en una variable y cómo se almacena y procesa. Los tipos de datos en Java pueden dividirse en dos categorías principales: tipos de datos primitivos y tipos de datos de referencia.

Existe una clasificación amplia respecto a los tipos de que se manejan en Java, sin embargo, podemos resumirla como se muestra en la figura 25.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 25
Tipos de datos en Java

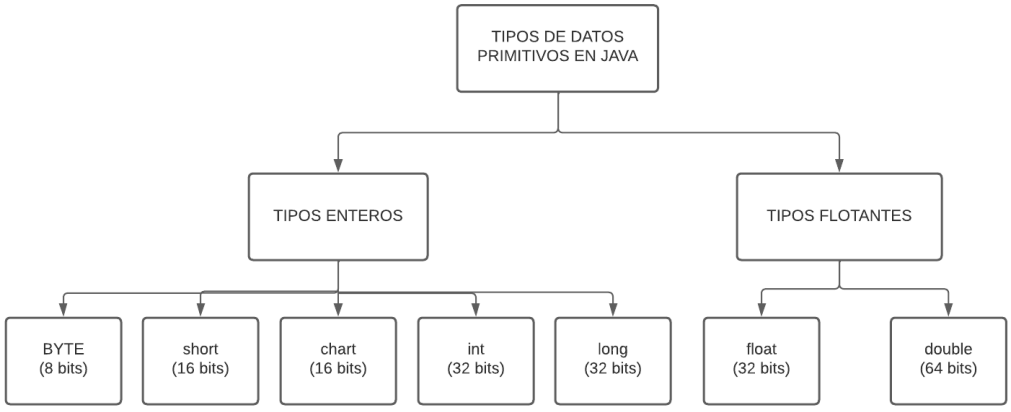


Nota: La imagen representa a los tipos de datos que se maneja en Java.

Por un lado, tenemos los tipos de datos primitivos y por otro lado tenemos los tipos que se consideran como extensiones de la clase Object, también conocidos como referencia de objetos.

Los tipos de datos primitivos se dividen en tipos de datos enteros y flotantes y estos se dividen en otros tipos de datos que lo mostraremos en la figura 26.

Figura 26
Tipos de datos primitivos en Java



Nota: La figura representa a los datos primitivo en Java.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Como se muestra en la figura anterior por un lado tenemos los tipos de datos enteros que se divide:

Tabla 2
Tipos de datos primitivos

TIPO DATOS	TAMAÑO
Byte	8 bits
Short	16 bits
Chart	16 bits
Int	32 bits
Long	64 bits

Nota: En la tabla se muestra los tipos de datos primitivos enteros con sus respectivos tamaños en bits.

Por otro lado, también tenemos los datos de tipo flotante que se divide en

Tabla 3
Tipos de datos Flotantes

TIPO DE DATO	TAMAÑO
Float	64 bits
Double	64 bits

Nota: En la tabla se muestra los tipos de datos primitivos flotantes con sus tamaños en bits

Estos tipos de datos en Java son los más básicos y son los que utilizaremos si necesitamos la mayor rapidez y ahorro en recurso, con el objetico de que nuestro programa sea más eficiente. Sin embargo, en la práctica también utilizaremos también funciones ya creadas que pueden hacer uso de manera indirecta de estos tipos de datos primitivos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.3.1. Tipos de Datos Enteros

Los tipos de datos enteros en Java son categorías que representan valores numéricos enteros, es decir, números sin parte decimal. Estos tipos permiten almacenar y manipular números enteros positivos y negativos en programas Java. Los tipos de datos enteros se utilizan para variables que necesitan contener números enteros, como recuentos, índices, edades y otros valores que no requieren fracciones o decimales. Algunos ejemplos de tipos de datos enteros en Java incluyen int, byte, short y long, cada uno de los cuales tiene un rango de valores que puede representar, lo que afecta a la cantidad de memoria que ocupan y su precisión en la representación de números enteros. Estos tipos de datos enteros son fundamentales en programación y se utilizan ampliamente en una variedad de aplicaciones Java.

3.3.2. Declaraciones de Variables

Una de las formas más simples de declarar variables en Java es primero se escribe el tipo de variable seguido por el nombre de la variable, si se desea asignar un valor inicial a la variable, y el formato de declaración es el siguiente.

```
<tipo de dato> <nombre de variable = <valor inicial>
```

También en Java se puede declarar múltiples variables en la misma línea siempre y cuando sean del mismo tipo de variable.

```
<tipo de dato> <nom_var1>,<nom_var2>...<nom_varn>
```

Ejemplo:

```
int valor;
int valor1=99;
int valor3, valor2;
int num_parte = 1233, num_items=23;
long num4, num5=991144222;
```

3.3.3. Tipos de Coma Flotante

Los tipos de datos de coma flotante o punto flotante como lo llaman representan números reales que contiene una coma o punto decimal, tal como 3.14159, o número grandes como $1.34 * 10^{15}$. La declaración de las variables de tipo flotante es igual a la declaración de las variables de tipo entero.

```
float valor;         //declara una variable real
float valor1=99.99f; //asigna 99.99 a la variable valor1
float valor3, valor2; //declara varios valores de coma flotante
double prod;
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Java soporta dos formatos de coma flotante: float, esta tipo de datos requiere 4 bytes de memoria mientras que el tipo double requiere 8 bytes.

3.3.4. Caracteres

Un carácter en Java se refiere a un tipo de dato que representa un solo símbolo o letra de un conjunto de caracteres. Este tipo de dato se utiliza para almacenar caracteres individuales, como letras, números, signos de puntuación y otros símbolos, en un programa Java. Los caracteres se representan mediante el tipo de dato *char* y pueden utilizarse para procesar y manipular texto en una aplicación Java.

3.3.5. Boolean

Un tipo de dato booleano en Java se refiere a una categoría que representa dos valores posibles: verdadero o falso. Este tipo de dato se utiliza para realizar evaluaciones lógicas y controlar el flujo de un programa. En Java, el tipo de dato booleano se declara con la palabra clave boolean y se utiliza para almacenar resultados de comparaciones o expresiones lógicas, como condiciones en sentencias if, while, for, entre otras. Los valores booleanos, verdadero o falso, son fundamentales para la toma de decisiones en la programación, ya que permiten controlar el comportamiento del programa en función de condiciones específicas.

```
boolean dulce;
dulce=true;
boolean encontrar, bandera;
```

3.4. Variables

Una variable en Java es un contenedor que se utiliza para almacenar y manipular datos en un programa. Cada variable tiene un nombre único y un tipo de dato que especifica qué tipo de información puede almacenar, como números, texto o valores lógicos. Las variables permiten a los programadores guardar y gestionar información en la memoria del programa, lo que facilita la creación de aplicaciones dinámicas y flexibles, por ejemplo.

```
char valor;
```

el ejemplo significa que el programa va a reservar un espacio de memoria para *valor*, en este caso el espacio de memoria reservado será de dos bytes. El identificador de la variable deber ser valido, y debe representar a lo que se está programando, por ejemplo:

```
sueldo      sueldo_diario      edad_empleado   $sueldo
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.4.1. Declaración

Cuando se declara una variable se necita una sentencia que proporciona información de esta al compilador, la sintaxis que se utiliza es la siguiente:

```
tipo variable
```

tipo es el nombre del tipo de dato que se va a utilizar
variable, es el identificador o nombre valido den Java.

Ejemplo:

```
int numero;
double sueldo_diario,
        horas_trabajadas,
        sueldo_semanal;
short diaSemana;
```

Antes de utilizar las variables es preciso declararlas, las declaraciones se les puede hacer en tres lugares del programa:

- En una clase, como miembro de esta
- Al principio de un método o un bloque de código
- En el lugar donde se va a utilizar

En una Clase

Las variables se declaran como un miembro de una clase esto quiere decir que estas variables se pueden utilizar en cualquier parte del programa.

```
public class Auto {
    double velocidad;
    void entrada(){
        velocidad=56.56;
    }
    void salida(){
        System.out.println("La velocidad el auto es: " + velocidad);
    }

}
```

En esta clase, la variable *velocidad* puede utilizarse en cualquier método, como en el ejemplo lo podemos utilizar en el método *entrada()* y *salida()*.

Al Principio de un Bloque de Código

Las variables también se pueden declarar en un método o un bloque de código dentro de un método.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
double velocidad(int n){
    int r;
    double c;
    c=1L;
    for (r = 1; r < n ; r++) {
        double aux=20;

    }

    return c;
}
```

En el ejemplo, las variables *r* y *c* están definidas dentro del método *velocidad* y estas variables son locales esto quiere decir que se pueden utilizar solo dentro del método, la variable *aux* esta declara dentro del ciclo *for* por la cual esta solo se le puede utilizar dentro de este.

En el Punto de Utilización

Java es un lenguaje de programación que proporciona una gran flexibilidad al momento de declarar variables, ya que se puede declarar variables donde se les va a utilizar, este tipo de declaraciones se utiliza mucho cuando se utiliza bucles.

```
for (int i = 0; i < 10; i++) {
    //...
}
```

En la sección del código se declara una variable dentro del ciclo *for* en la cual hace un recorrido desde el 0 hasta el 10 y se va aumentando de uno en uno.

Inicialización de Variables

Al momento de declarar una variable esta no tiene valor lo que podemos hacer es darle un valor inicial cuya sintaxis es la siguiente.

```
tipo nombre_variable= expresion;
```

expresión, es cualquier declaración valida cuyo valor es el mismo modelo del *tipo,* por ejemplo.

```
char genero = 'F';
int acumulador= 0;
int anio=2023;
```

En el ejemplo anterior vemos que al crear las variables se les da un valor, que se guardan en la memoria, una variable inicializada o no pueden cambiar de valor utilizando sentencias de asignación como, por ejemplo:

```
int anio;
anio=2024;
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.5. Duración de una Variable

Dependiendo donde se usa las variables de Java, estas se pueden utilizar en todo el programa, dentro de un método o dentro de un bloque de código de manera temporal, el lugar del programa donde se activa la variable se llama alcance o (scope), por lo general este alcance se extiende desde donde se define la variable hasta el final del bloque, de acuerdo con esto existen dos tipos de alcen que son:

- Variables Locales
- Variables de clase

3.5.1. Variables Locales

Estas variables son definidas dentro de un método, estas solo están disponible o visibles dentro de esa sección del programa o método especifico, para este tipo de variables rige algunas reglas que son.

1. En el interior del método no se pueden modificar por ninguna sentencia externa a aquel.
2. Sus nombres no son únicos, eso quiere decir que dos o más métodos pueden declarar variables con el mismo nombre, y cada una de ellas son distintas y solo pertenecen a ese método.
3. Las variables no existen en memoria hasta que se ejecute el método donde están declaradas, de esta forma me permite ahorrar memoria ya que deja que varios métodos compartan la misma memoria para sus variables locales, pero no de manera simultánea.

Como ya hemos mencionado por esta última razón las variables locales también se llaman automáticas ya que se crean de manera predeterminada al inicio del método y termina cuando finaliza la ejecución del método.

```
public static void sumar(){
    double n1,n2,n3;
    //declaramos una variable local
    int resultado;
    n1=Math.random()*999;
    n2=Math.random()*999;
    n3=Math.random()*999;
    resultado= (int) ((int)n1+n2+n3);
    System.out.println("La suma de los números aleatorios son " +
resultado);

}
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

3.5.1. Variables de Clases

Sabemos que los miembros de una clase son los métodos y variables, cuando se declara un variable fuera de los métodos estas están disponible y visibles para cualquier método, y se lo utiliza o referencia solo con su nombre.

```
public class Auto {

    //declramos variables globales o variables
    //de la clase
    double precio, cilindraje, interes;

    double costo()
    {
        //declaración de variables locales
        double x;
        precio= 25.000;
        return precio;
    }
    //declaración de variable local
    double valor;
    double cilindraje(){
        //inicializamos la variable que solo se puede utilizar en esta
clase
        float x;
        //esta variable es visible porque esta declarado en la clase
        cilindraje=1.8;
        //inicializamos la variable valor que solo se puede
        //visualizar en esta clase
        valor = 0;
        valor = valor+precio;
        return valor;
    }


}
```

3.6. Entrada y Salida

La entrada y salida de datos en Java se refiere al proceso de trasferir información hacia y desde un programa de Java. Esto permite que los programas interactúen con los usuarios o con otros sistemas para recibir datos y enviar resultados.

La clase *System* define dos referencias a objetos *static* para la gestión de entrada y salida por consola, estos son:

```
System.in    //para la entrada por teclado
System.out   //para la salida por pantalla
```

En Java *System.in* es una entrada estándar predefinida que representa el flujo de entrada de datos desde el teclado o desde otro dispositivo de entrada. Es una instancia de la clase *InputStream* y se utiliza para recibir información introducida por el usuario durante la ejecución de un programa. Esta entrada estándar es ampliamente utilizada para interactuar con el usuario y recopilar datos en tiempo real en aplicaciones de consola,

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

además hace referencia a un objeto de la clase *BufferedInputStream* en la cual hay diversos métodos para captar caracteres tecleados.

La segunda referencia a un flujo de salida estándar predefinido que representa la forma en que los programas pueden mostrar información y resultados al usuario. Se trata de una instancia de la clase *PrintStream* y cumple un rol fundamental en la comunicación entre el programa y el usuario en aplicaciones de consola.

Este mecanismo permite que los programas emitan mensajes, datos y resultados de cálculos directamente en la pantalla del usuario. Es ampliamente utilizado para mostrar mensajes informativos, resultados de operaciones, mensajes de error o cualquier otro tipo de salida que sea relevante para el usuario.

Cuando se emplea *System.out*, el programa escribe datos en forma de secuencias de caracteres, lo que facilita la presentación de información de manera legible y comprensible para el usuario. Puede utilizarse para imprimir texto, valores numéricos, datos procesados o cualquier información relevante para el flujo de trabajo de la aplicación.

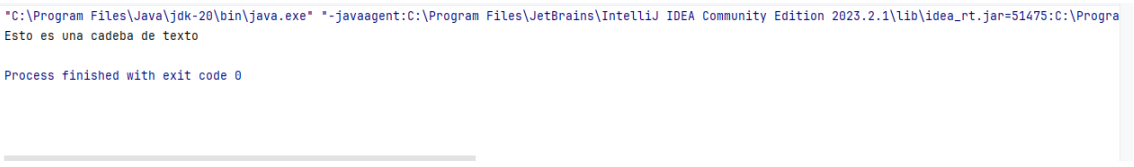
Es importante mencionar que *System.out,* está vinculado a la consola por defecto, pero se puede redirigir hacia otros destinos de salida, como archivos o dispositivos externos, para lograr una mayor flexibilidad en la gestión de la información generada por el programa.

3.6.1. Salida (System.out)

Cuando se quiere imprimir un dato en consola Java tiene un objeto *out* que ese encuentra definido en la clase *System,* y permite visualizar los datos en la pantalla de su equipo, por ejemplo:

```
public static void main(String[] args) {
    System.out.println("Esto es una cadeba de texto");
}
```

Figura 27
Salida de información en consola



Nota: La imagen representa la salida de pantalla del código donde muestra el mensaje “Eso es una cadena de texto”. Captura de pantalla

System.out es una referencia al objeto de la clase *PrintStream,* este objeto tiene diferentes métodos los cuales lo podemos utilizar con frecuencia.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
print()  //Transfiere una cadena de caracteres al buffer de la
pantalla
println() //transfiere uan cadena de caracteres y el caracter de fin
de línea al buffere de pantalla
flush() //el buffer con las cadenas almacenadas se imprime en pantalla
```

3.6.2. Entrada

La clase System define un objeto de la clase BufferedInputStream cuya referencia resulta en in. El objeto se asocia al flujo estándar de entrada, que por defecto es el teclado, los elementos básicos de este flujo son caracteres individuales y no cadenas como ocurre con el objeto out; entre los métodos de la clase se encuentra read() que devuelve el carácter actual en el buffer de entrada; por ejemplo:

Para realizar entrada de datos por teclado con la clase BuffereReader se importa la clase

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
```

importada la clase lo que se hace es crear un objeto de esta clase BufferedReader y de la clase *InputStreamReader*

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

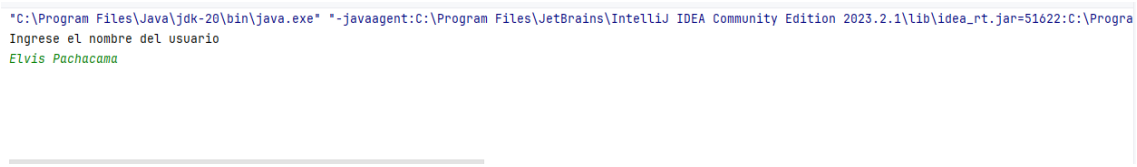
Inicializados los objetos *InputStreamReader* y *BufferedReader* nos permite capturar líneas de caracteres mediante el teclado con el método *readLine()*.

```
package vehiculo;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Auto {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String nombre;
        System.out.println("Ingrese el nombre del usuario");
        nombre = br.readLine();
    }

}
```

Como podeos ver en el código anterior además de las dos clases importadas, se debe importar un control de excepciones para el manejo de caracteres *IOException*.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 28
Salida en consola



Nota: La figura representa la salida de nuestro código en consola donde pide que se ingrese el nombre del usuario por teclado. Captura de pantalla

Cuando se ingresa datos por consola como vemos en el ejemplo del código anterior, no solo se puede ingresar cadena de caracteres, sino Java permite ingresar cualquier tipo de datos, por ejemplo:

Ejemplo 3.3

Crear un programa en Java donde el usuario ingrese su nombre completo y la edad en números enteros.

```
package nombre;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
/*
* Programa: Alumno.java
* Programador: Ing. Elvis Pachacama
* Descripción. Crear un programa donde el usuario ingrese
*              por teclado el nombre del alumno y la edad los tipos de
*              ingresar son nombre tipo String, edad tipo entero
* Fecha de creación: 26 de septiembre del 2023
* Revisón: ninguna*
* */
public class Alumno {
    //método main
    public static void main(String[] args) throws IOException {
        //Inicializamos el objeto de la clase BufferedREader y de la
clase InputStreamReader
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        //declaramos las variables a utilizar nombre de tipo String y
edad de tipo entero
        String nombre;
        int edad;
        //Imprimimos un mensaje en consola pidiendo al usuriao
ingresar el nombre
        System.out.println("Ingrese el nombre del Alumno:");
        //Ingresamos y guardamos en la variable nombre lo que ingrese
el usuario
        nombre=br.readLine();
        //Imprimimos un menaje en consola pidiendo al usuario ingrese
al edad del Alumno
        System.out.println("Ingrese la edad delLAumno:");
        //Ingresamos y transformamos a entero la edad
        edad = Integer.parseInt(br.readLine());
    }
}
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Algo para recordar cuando usamos la clase *BufferedReader*, todos los datos que se ingresan lo hacen de tipo *String*, como se muestra en el código anterior, se ingresa por teclado y se guarda en una variable de tipo *int* la eda del alumno, para eso debemos convertir esa cadena de tipo *string* a *entero*, y de esa manera se puede ingresar cualquier tipo de dato.

Tabla 4
Tipo de conversión de datos

Tipo dato	Conversión
int edad	edad=Integer.*parseInt*(br.readLine());
double sueldo	`sueldo= Double.parseDouble(br.readLine());`
float sueldo	`sueldo= Float.parseFloat(br.readLine());`

Nota: La tabla muestra como convertir un dato en Java cuando se utiliza la clase BufferedReader

3.6.3. Entrada con la Clase (*Scanner)*

La entrada de datos con la clase Scanner en Java es una técnica comúnmente utilizada para obtener información ingresada por el usuario a través del teclado o para leer datos desde otros flujos de entrada, como archivos. La clase Scanner se encuentra en el paquete java.util y proporciona métodos para analizar y procesar diferentes tipos de datos de entrada, como números enteros, números de punto flotante, cadenas de caracteres y más. Aquí te muestro un ejemplo de cómo usar la clase Scanner para obtener entrada de datos desde el teclado:

```
package nombre;
import java.util.Scanner;
/*
* Programa: Alumno.java
* Programador: Ing. Elvis Pachacama
* Descripción. Crear un programa donde el usuario ingrese
*              por teclado el nombre del alumno y la edad los tipos de
*              ingresar son nombre tipo String, edad tipo entero
* Fecha de creación: 26 de septiembre del 2023
* Revisón: ninguna*
* */
public class Alumno {
    //método main
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
    public static void main(String[] args) {
        //instanciamos el objeto de la clse Scanner
        Scanner entrada = new Scanner(System.in);
        //imprimimos un mensaje en la pamtalla para que el usuario
ingrese
        //el nombre del alumno
        String nombre;
        int edad;
        double matricula;
        System.out.println("Ingrese el nombre del Alumno: ");
        nombre = entrada.nextLine();
        System.out.println("Ingrese la edad del Alumno");
        edad=entrada.nextInt();
        System.out.println("Ingrese el valor de la matricula");
        matricula = entrada.nextDouble();

    }
}
```

Una vez que se crea el objeto *Scanner,* se puede utilizar diferentes métodos para leer la entrada de datos: *nextInt* o *nextDouble,* leen datos de tipo enteros y de coma flotante.

Cuando el compilador llama a uno de los métodos anteriores, el programa espera hasta que el usuario teclee un número y pulse Enter.

El método *nextLine()* lo que hace es leer una línea de entrada, cuando se utiliza el método *next()* este método sirve para leer una palabra sin espacios.

En Java para usar la clase *Scanner* se debe importar el paquete *java.util* y siempre que se utiliza una clase que no está definida en el paquete básico *java.lang* se necita utilizar la directiva *import*. La primera línea del código indica a Java dónde encontrar la definición de la clase *Scanner*.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen del Capítulo 3

Este capítulo nos introdujo a los componentes básicos de un programa en Java, en los siguientes capítulos se analizará con detalle cada uno de estos componentes, de igual manera se analizó el modo de utilizar las características mejoradas en Java que nos permitirá escribir programas orientados a objetos. En este capítulo además de aprender la estructura general de un programa en Java y que:

- **Main:** Cuando se ejecuta un programa en Java, el sistema comienza por buscar y ejecutar el método *main* en la clase principal. Todo el código que desees que se ejecute de residir dentro de este método.
- **Métodos definidos por el usuario.** Los métodos definidos por el usuario son una característica fundamental de Java que mejora la estructura y eficiencia del código.
- **Paquetes.** Los paquetes en Java son una característica fundamental que facilita la organización y la gestión de código en proyectos de programación Java, mejorando la claridad, la reutilización y el modularidad del código.
- **Tipos de datos enteros.** Un tipo de dato entero se utiliza para representar números enteros sin parte decimal, como enteros positivos y negativos.
- **Tipos de datos flotantes.** Los datos de coma flotante se utilizan para representar números con decimales, como números reales o fracciones y puede incluir *float y double.*
- **Caracteres.** Un carácter es una representación de un símbolo o letra individual, y el tipo de datos *char* se utiliza para almacenar y trabajar con este tipo de caracteres en aplicaciones Java
- **Boolenanos.** En Java un tipo de dato booleano es una categoría que representa valores lógicos de verdadero o falso y se utiliza para realizar evaluaciones lógicas y controlar el flujo de un programa
- **Entrada y salida de datos.** La entrada y salida de datos son componentes cruciales en la mayoría de las aplicaciones Java, ya que permiten que los programas interactúen con los usuarios y con otros sistemas de manera efectiva. Java ofrece una variedad de clases y métodos que facilitan estas operaciones, lo que hace que sea posible crear aplicaciones interactivas y que puedan leer y escribir datos de manera eficiente.
- **System.in,** en Java es el mecanismo que permite la entrada de datos desde el teclado o dispositivos de entrada estándar en tiempo real

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

- **System.out,** en Java es un componente esencial para la salida de datos en aplicaciones de consola, permitiendo que los programas muestren información y resultados al usuario de manera efectiva y legible

Ejercicios

3.1. ¿Cuál es la salida del siguiente programa?

```
public class Alumno {
    //método main
    public static void main(String[] args) {
        System.out.println("Hola Mundo desde Java!\n" + "Programando
soluciones");

    }
}
```

3.7. Identificar el error del siguiente programa

```
public class Alumno {
    //método main
    main() {
        System.out.println("Hola mundo!\n");

    }
}
```

3.8. Escribir un programa donde el usuario ingrese por teclado el nombre y la dirección de una persona y la imprima en consola, el programador debe utilizar la clase *Scanner*. Para resolver el problema.

3.9. Escribir un programa donde el usuario ingrese los datos de la compra de un vehículo utilizando la clase *InputStreamReader* e imprimirlos en pantalla, los datos a ingresar son:

- Nombre del comprador.
- CI
- Dirección
- Teléfono
- Correo electrónico
- Precio del vehículo
- Valor del Iva 12%

Los datos ingresados deben ser en el formato que se los pide.

3.10. Escribir un programa en Java que imprima el mapa de tu país con asteriscos.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Capítulo 4

Operadores y Expresiones

4.1. Operadores y Expresiones

En Java, los operadores y las expresiones son elementos fundamentales para realizar cálculos, tomar decisiones y realizar diversas operaciones en un programa.

Los programas Java constan de datos, declaraciones y expresiones; estas últimas suelen representar ecuaciones matemáticas; por ejemplo: 7*x 2*z; donde los símbolos más y de producto (+, *) son operadores de suma y de producto; los números 7 y 2 y las variables x y z son operandos; En pocas palabras, una expresión es una secuencia de operaciones y operandos que especifican un cálculo.

Cuando se usa entre números o variables, este operador se llama operador binario porque suma dos números. Otro tipo de operador en Java es el operador unitario o *unario*, que opera sobre un único valor; si la variable x es igual a 5, entonces -x será igual a= -5; el signo menos (-) representa la resta del operador de valor unitario. Java admite un potente conjunto de operadores unarios, binarios y otros.

Sintaxis

```
variable=expresión
```

variable identificador valido para Java, que es declarado como variable

expresión constante, otra variable con un valor previamente asignado o una fórmula evaluada de tipo *variable*.

4.2. Operadores de Asignación

Los operadores de asignación en Java son símbolos especiales utilizados para asignar un valor a una variable. Estos operadores desempeñan un papel fundamental en la manipulación y gestión de datos en programas escritos en Java. A través de estos operadores, podemos actualizar el contenido de variables de manera eficiente y concisa.

El operador de asignación básico es el signo igual (=). Su función principal es asignar el valor a la derecha del operador al nombre de variable a la izquierda. Por ejemplo:

```
int numero = 23;
```

Como se ve en el ejemplo el signo = esta asignando un valor a la variable número en el que se guarda el 23, este valor se puede cambia. Sin embargo, Java también ofrece

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

operadores compuestos de asignación que combinan una operación con la asignación en una sola expresión. Estos operadores compuestos permiten realizar operaciones aritméticas y luego asignar el resultado a la variable en un solo paso. Los operadores compuestos de asignación incluyen:

- : Suma y asignación. Por ejemplo, x += 5; es equivalente a x = x + 5;.
- -=: Resta y asignación. Por ejemplo, y -= 3; es equivalente a y = y - 3;.
- *=: Multiplicación y asignación. Por ejemplo, z *= 2; es equivalente a z = z * 2;.
- /=: División y asignación. Por ejemplo, w /= 4; es equivalente a w = w / 4;.

Estos operadores compuestos son especialmente útiles cuando se trabaja con variables y se necesita actualizar su valor en función de operaciones previas.

4.3. Operadores Aritméticos

Los operadores aritméticos en Java son símbolos que se utilizan para realizar operaciones matemáticas en valores numéricos. Estos operadores permiten realizar cálculos como suma, resta, multiplicación, división y módulo. A continuación, se proporciona una definición más detallada de los operadores aritméticos en Java:

- **Suma (+).** El operador de suma se utiliza para agregar dos valores numéricos y producir un resultado que es la suma de ambos. Por ejemplo:

```
int resultadoSuma = 23+25; //El resultado sera 48
```

- **Resta (-).** El operador de resta se utiliza para restar un valor numérico de otro y producir un resultado que es la diferencia entre ellos. Por ejemplo:

```
int resultadoResta=65-34;
```

- **Multiplicación (*).** El operador de multiplicación se utiliza para multiplicar dos valores numéricos y producir un resultado que es el producto de ambos. Por ejemplo:

```
int resultadoMultiplicacion=5*23;
```

- **División (/).** El operador de división se utiliza para dividir un valor numérico por otro y producir un resultado que es el cociente de la división. Es importante notar que, si los operandos son números enteros, la división se realizará como una división entera, lo que significa que se truncará cualquier parte decimal.

```
double resultadoDivision=34/6;
```

- **Módulo (%).** El operador de módulo se utiliza para encontrar el resto de la división entre dos números enteros. Es útil para verificar si un número es

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

divisible por otro y para realizar operaciones relacionadas con ciclos y patrones repetitivos.

```
double resultadoModulo= 10%3; // el módulo es 1
```

4.4. Operados de Incremento y Decremento

Los operadores de incremento y decremento en Java son operadores unarios que se utilizan para aumentar o disminuir el valor de una variable en una unidad. Estos operadores son extremadamente útiles en bucles y en situaciones donde es necesario actualizar una variable de manera iterativa. A continuación, se proporciona una definición de los operadores de incremento y decremento en Java:

1. **Operador de Incremento (++):** El operador de incremento (++), a veces llamado "incremento de uno", se utiliza para aumentar el valor de una variable en una unidad. Puede aplicarse a variables numéricas (como enteros o números de punto flotante) y también a variables de tipo char, que representan caracteres.
    a. **Prefijo (++variable):** Cuando se utiliza el operador de incremento en forma de prefijo, primero se incrementa el valor de la variable y luego se utiliza ese valor en la expresión actual.

    ```
    int contador =6;
    int resultado = ++contador;//Incrementamos al contador más 1 y el resultado sera 7
    ```

    b. **Postfijo (variable++):** Cuando se utiliza el operador de incremento en forma de postfijo, primero se utiliza el valor actual de la variable en la expresión actual y luego se incrementa.

    ```
    int contador =6;
    int resultado = contador++;//Utiliza el valor actual del contador que 6 luego incrementa el contador en 1
    ```

2. **Operador de Decremento (--):** El operador de decremento (--), similar al operador de incremento, se utiliza para disminuir el valor de una variable en una unidad. Al igual que con el operador de incremento, el operador de decremento puede aplicarse en forma de prefijo o postfijo.
    a. **Prefijo (--variable):** En esta forma, primero se decrementa el valor de la variable y luego se utiliza ese valor en la expresión actual.

    ```
    int contador =6;
    int resultado = --contador;//Decrementa el contador en 1 y luego lo asigana a resultado que es 5
    ```

    b. **Postfijo (variable--):** En esta forma, primero se utiliza el valor actual de la variable en la expresión actual y luego se decrementa.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

```
int contador =6;
int resultado = contador--;//Utiliza el contador con el
valor actual en resultado y luego lo decrementa en 1 valor
5
```

4.5. Operados Relacionales

Debido a que el tipo de datos booleano de Java tiene los valores falso y verdadero, una expresión booleana es una secuencia de operandos y operadores que se combinan para producir uno de sus valores posibles.

Los operadores que prueban la relación entre dos operandos (como >= o ==) se denominan operadores relacionales y se utilizan en expresiones de la siguiente forma:

Los operadores relacionales se utilizan normalmente en declaraciones selección (if) o las iteraciones (while, for) se utilizan para probar condiciones; a través de ellos se realizan las operaciones de igualdad, desigualdad y diferencia relativa; la tabla 5 enumera los operadores relacionales que se pueden aplicar a cualquier operando de tipo de datos estándar: char, int, float, double, etc.; cuando se usa en una expresión, se evalúa como verdadero o falso según el resultado de la condición; Por ejemplo.

```
boolean c;
c= 4<8;
```

la variable c se asigna a verdadero (true), puesto que 4 es menor que 8, entonces la operación < devuelve un valor verdadero (true), que se asigna a c

Tabla 5
Operadores relacionales en Java

Operador	Significado	Ejemplo
==	Igual a	x==z
¡=	No igual a	A ¡=b
>	Mayor a	a>b
<	Menor a	B<a
>=	Mayor igual	a>=b
<=	Menor igual	C<=b

Nota: La tabla representa los operadores relacionales que se utiliza en Java con su significado y ejemplos

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

4.6. Operadores Lógicos

Además de operadores matemáticos, Java tiene operadores lógicos: if, while o for, ya mencionados que se estudiarán más adelante; éstos, denominados booleanos, en honor a George Boole, creador del álgebra que lleva su apellido, son: not (!), and (&&), or (||) y or exclusivo (^). not produce falso si su operando es verdadero y viceversa; and resulta en verdadero sólo si ambos operandos son verdaderos; en caso contrario, deriva en falso; or produce verdadero si cualquiera de los operandos tiene ese valor y resulta en falso sólo si ambos lo son; or exclusivo deriva en verdadero si ambos operandos son distintos: verdadero-falso o falso-verdadero, y produce falso sólo si ambos operandos son iguales: verdadero-verdadero o falso-falso. Además, hay que considerar que Java permite utilizar &, | como operadores and y or respectivamente, con el significado mencionado, salvo la evaluación en cortocircuito; la tabla 6 muestra los operadores lógicos de Java

Tabla 6
Operadores lógicos

Operador	Operación lógica	Ejemplo
Negación(!)	No lógica	!(x>=y)
o, exclusiva(^)	Operando_1 ^Operando_2	X<n ^ n>9
Y, lógica (&&)	Operando_1 && Operando_2	m<n&&k>l
O, lógica ||	Operando1||operando2	m=5||n!=10

Nota: La tabla muestra los operadores lógicos que se utilizan en las condicionales con sus respectivos ejemplos

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen del Capítulo 4

En este capítulo se examinó los diferentes tipos de operadores que utiliza Java y cual es su sintaxis y ejemplos.

- **Los operadores en Java**, son herramientas esenciales para realizar operaciones matemáticas, comparar valores, tomar decisiones lógicas y controlar el flujo de un programa. Combinados con variables y estructuras de control, los operadores permiten crear programas funcionales y eficientes en Java. Su comprensión y uso adecuado son fundamentales para cualquier programador de Java.
- **Operadores de asignación,** en Java son elementos esenciales para asignar valores a variables y actualizarlas de manera eficiente
- **Operadores aritméticos,** Estos operadores aritméticos en Java son fundamentales para realizar cálculos matemáticos en programas y son ampliamente utilizados en aplicaciones que involucran manipulación de datos numéricos.
- **Operadores de incremento y decremento,** estos operadores son muy útiles para realizar tareas que involucran recorridos y manipulación de índices en arreglos y colecciones, así como para llevar un seguimiento de valores en bucles y otras estructuras de control.
- **Operadores relacionales,** Estos operadores de relación son fundamentales para la programación en Java, ya que permiten tomar decisiones basadas en comparaciones de valores.

Ejercicios

4.7. Determinar el valor de las siguientes expresiones aritméticas.

14/12

24/14

123/34

200/50

4.8. Escribir un programa en Java que lea por teclado dos números enteros de tres dígitos calcule e imprima producto, cociente y resto cuando el primero se divide entre el segundo.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

4.9. Escribir un programa en Java donde el usuario ingrese la temperatura en Celcius por teclado y convertir a Farenheit utilizando la siguiente formula.

$$f = \frac{9}{5}C + 32$$

4.10. Escribir un programa en Java para obtener la Hipotenusa y los ángulos de un triángulo rectángulo a partir de las longitudes de los catetos, todos los datos se ingresan por teclado.

4.11. Escribir un programa en Java donde el Usuario ingrese un número por teclado y verifique si el número es primo o no.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Capítulo 5

Estructuras de Selección

5.1. Estructuras de Control

Las estructuras de control en Java son herramientas esenciales que permiten a los programadores administrar cómo se ejecutan las instrucciones en un programa Java. Estas estructuras permiten tomar decisiones, repetir acciones y controlar el flujo de ejecución de manera que las instrucciones se ejecuten en el orden y bajo las condiciones deseadas. En esencia, las estructuras de control en Java son elementos cruciales que brindan flexibilidad y lógica a la programación, permitiendo la automatización de tareas y la toma de decisiones basadas en condiciones específicas.

```
{
    senticia 1;
    sentencia 2,
    .
    .
    .
    sentencia n
}
```

El flujo de control va de la declaración 1 a la declaración 2 y así sucesivamente; sin embargo, algunos problemas requieren pasos con dos o más opciones o alternativas elegidas en función del valor de una condición o expresión.

5.1.1. Sentecia if

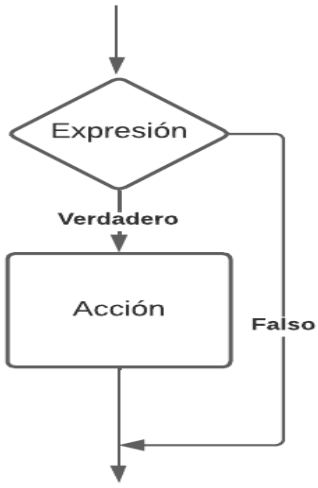
La principal estructura de control de selección de Java es la declaración if; tiene dos alternativas o formatos posibles, el más simple tiene la siguiente sintaxis:

```
if(expresion) Acción
```

La declaración if funciona así: cuando se alcanza, se evalúa la siguiente expresión entre paréntesis; si la expresión es verdadera, la acción se ejecuta, de lo contrario no se ejecuta y la ejecución continúa con la siguiente declaración. Acción es una sentencia simple o compuesta, en la figura xx muestra un diagrama de flujo que indica el proceso de ejecución del programa.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 29
Diagrama de flujo de una sentencia básica if



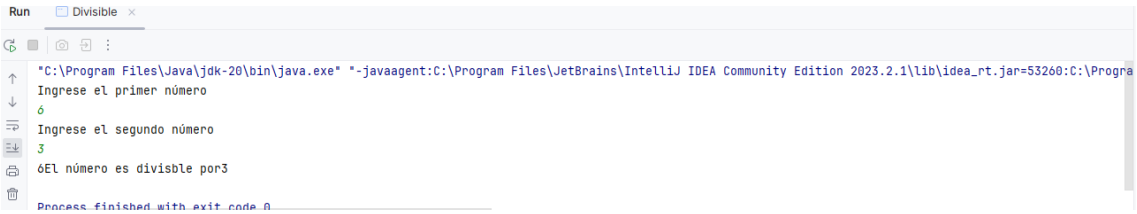
Nota. La figura representa el diagrama de flujo de una sentencia básica

Ejemplo 5.1

Escribir un programa en Java donde el usuario ingrese dos números enteros por teclado y determinar si el primer número es divisible para el segundo número.

```
package divisible;

import java.util.Scanner;

public class Divisible {
    public static void main(String[] args) {
        int n,d;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese el primer número");
        n=entrada.nextInt();
        System.out.println("Ingrese el segundo número");
        d=entrada.nextInt();
        if(n%d==0){
            System.out.println(n + "El número es divisble por" + d);
        }
    }
}
```

Figura 30
Ejecución del programa



Nota: La imagen muestra la ejecución del código anterior donde el primer número es divisible para el segundo número ingresado.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

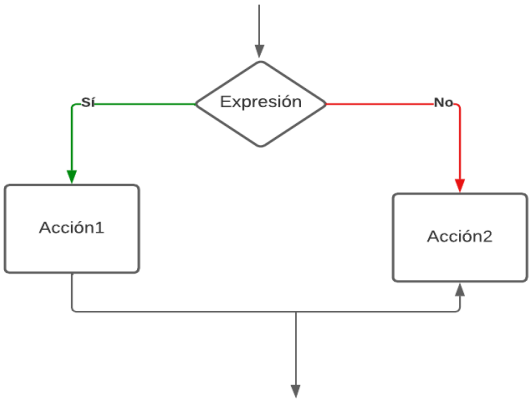
5.1.2. Sentecia if-else

Un segundo formato del if es la expresión if-else, cuyo formato tiene la siguiente sintaxis:

```
if(expresion)
    accion1
else
    accion2
```

En este formato acción 1 y acción 2 son, de forma individual, una única sentencia que termina en un punto y coma, o un grupo de sentencias entre llaves; expresión se evalúa cuando se ejecuta la sentencia: si es verdadera, se efectúa acción 1; en caso contrario se ejecuta acción 2, la figura 5.2 muestra su semántica.

Figura 31
Diagrama de flujo sentencia if-else



Nota: La figura representa el diagrama de flujo de la sentencia if-else

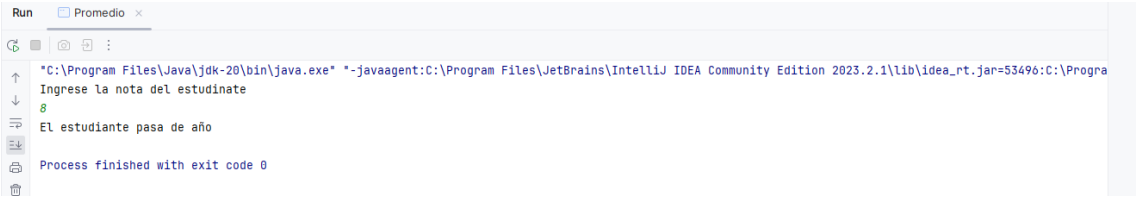
Ejemplo 5.2

Escribir un programa donde el usuario ingrese una nota del examen, si la nota del examen es mayor a 7 mostrar un mensaje pasa de año, caso contrario si la nota es menor igual a 7 mostrar el mensaje el estudiante pierde el año

```
package notas;
import java.util.Scanner;
public class Promedio {
    public static void main(String[] args) {
        int nota;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese la nota del estudinate");
        nota=entrada.nextInt();
        if (nota > 7) {
            System.out.println("El estudiante pasa de año");
        }else{
            System.out.println("El estudiante no pasa el año");
        }
    }
}
```


(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Figura 32
Salida del programa if-else



Nota: La figura muestra la salida en pantalla del programa de notas, donde si el estudiante saca más de 7 pasa de año, caso contrario pierde.

5.1.3. Sentencia de Control Switch

En Java, switch es una sentencia que se utiliza para elegir una de entre múltiples opciones; es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada expresión de control o selector; el valor de dicha expresión puede ser de tipo int o char pero no de tipo double.

Sintaxis

```
switch(selector)
{
    case etiquea1 : sentencia1;
        break;
    case etiqueta2 : setencia2;
        break;
    .
    .
    .
    case etiquetan : sentenciasn;
        break;
    default: sentenciasd ;
}
```

a expresión de control o selector se evalúa y compara con cada una de las etiquetas de case; además, debe ser un tipo ordinal, por ejemplo, int, char, bool pero no float o string; cada etiqueta es un valor único, constante y debe tener un valor diferente de los otros. Si el valor de la expresión es igual a una de las etiquetas case, por ejemplo, etiqueta1, entonces la ejecución comenzará con la primera sentencia de sentencias1 y continuará hasta encontrar break o el final de switch. El tipo de cada etiqueta debe ser el mismo que la expresión de selector; las expresiones están permitidas como etiquetas, pero sólo si cada operando de la expresión es por sí misma una constante; por ejemplo, 4, +8 o m*15, siempre que m hubiera sido definido anteriormente como constante nombrada. Si el valor del selector no está listado en ninguna etiqueta case no se ejecutará ninguna de las opciones a menos que se especifique una acción predeterminada. La omisión de una etiqueta default puede crear un error lógico difícil de prever; aunque ésta es opcional, se

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

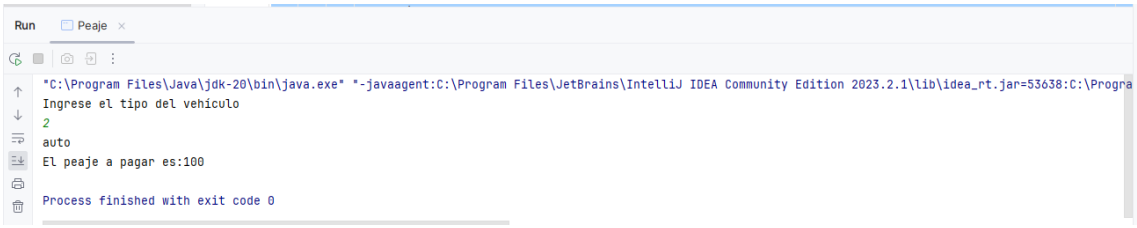
recomienda su uso, a menos de estar absolutamente seguro de que todos los valores de selector están incluidos en las etiquetas case.

Ejemplo 5.3

Escribir un programa donde el programa simule un peaje donde ingrese el tipo de vehículo y me muestre el valor de pagar en el peaje.

```
package peaje;

import java.util.Scanner;

public class Peaje {
    public static void main(String[] args) {
        int tipoVehiculo, peaje;
        Scanner entrada= new Scanner(System.in);
        System.out.println("Ingrese el tipo del vehículo ");
        tipoVehiculo=entrada.nextInt();
        switch (tipoVehiculo){
            case 1:
                System.out.println("Pesado");
                peaje=500;
                System.out.println("El peaje a pagar es:" + peaje);
                break;
            case 2:
                System.out.println("auto");
                peaje=100;
                System.out.println("El peaje a pagar es:" + peaje);
                break;
            case 3:
                System.out.println("moto");
                peaje=50;
                System.out.println("El peaje a pagar es:" + peaje);
                break;
            default:
                System.out.println("Opción no encontrada");
        }
    }
}
```

Figura 33
Salida de pantalla peaje



Nota: La figura representa a la salida de pantalla donde el usuario ingresa el tipo de vehículo y muestra cuanto debe pagar.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Resumen del Capítulo 5

Estructuras de Control en Java es fundamental para comprender cómo se diseña la lógica de un programa Java. Permite a los programadores tomar decisiones, crear aplicaciones interactivas y controlar el flujo de ejecución de manera efectiva. La comprensión y el dominio de estas estructuras son esenciales para escribir programas funcionales y lógicos en el lenguaje de programación Java.

- **Estructuras de control,** Las estructuras de control en Java son esenciales para escribir programas complejos y lógicos. Permiten que los programas realicen cálculos, tomen decisiones basadas en condiciones y ejecuten acciones de manera eficiente.
- **If,** es una herramienta fundamental en programación que permite realizar bifurcaciones en la ejecución del programa, ejecutando ciertas instrucciones solo cuando se cumple una condición dada.
- **If-else,** es una herramienta fundamental en programación que permite gestionar el flujo del programa, ejecutando diferentes acciones en función de si una condición es verdadera o falsa.
- **Switch,** es útil cuando se necesita tomar decisiones basadas en múltiples opciones y es una alternativa a las estructuras if y else if cuando se tienen varios casos para evaluar.

Ejercicios

5.1. ¿Cuáles son los errores de sintaxis que tiene la siguiente sentencia?

```
if x> 26.0
    y=x
else
    y=z;
```

5.2. Escribir un programa que determine si un año es bisiesto, para este programa se sabe que cuando es múltiplo de 4, por ejemplo, 1984, sin embargo, los años que son múltiplos de 100 sólo son bisiestos cuando también son múltiplos de 400, por ejemplo 1800 no es bisiesto, mientras que 2000, si lo es.

5.3. Escribir un programa en Java que resuelva la ecuación cuadrática ($ax^2 + bx + c = 0$)

5.4. Escribir un programa que calcule el número sde días de un mes, dados los valores numéricos del mes y el año.

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC

Referencias

ARNOLD, K., GOSLING, J. y HOLMES, D. (2007), The Java™ Programming Language, 4a. ed., Sun Microsystems

Caules, C. Á. (2023). Las versiones de Java y su historia. Arquitectura Java. https://www.arquitecturajava.com/las-versiones-de-java/

CADENHEAD, R. y LEMAY, L. (2007), Teach Yourself Java 6 in 21 Days, Indianápolis: Sams

DEITEL, H. M. y DEITEL, P. J. (2002), Java. How to Program, 4a. ed., Upper Saddle River, NJ: Prentice-Hall

Gardiner, J. M., & Java, R. I. (1993). Recognition memory and awareness: An experiential approach. *European Journal of Cognitive Psychology*, *5*(3), 337-346.

Goling, J., Joy, B., & Steele, G. (s.f.). The Java Programming Languaje.

Hennessy, J. L., & Paterson, D. A. (2019). Arquitectura de Computadoras: Un enfoque Cuantitativo. Pearson Education.

Joyanes Aguilar, L., & Zahonero Martinez, I. (2011). Programación en Java 6. Mexico: McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.

Palabras reservadas Java. (n.d.). CódigoFacilito. https://codigofacilito.com/articulos/palabras_reservadas_java

SAVITCH, W. (2008), Absolute Java, 3a. ed., Boston: Pearson/Addison-Wesle

Toro, J. (09 de 10 de 2018). Tipos de Progrmas que se pueden hacer en Java. Obtenido de Applets: http://joseltoro.blogspot.com/2018/10/que-tipos-de-programas-se-pueden-hacer.html

(+593) 96 356 1961 admisiones@itq.edu.ec Antonio de Ulloa N28-30 y Diego de Atienza (Esq). WWW.ITQ.EDU.EC